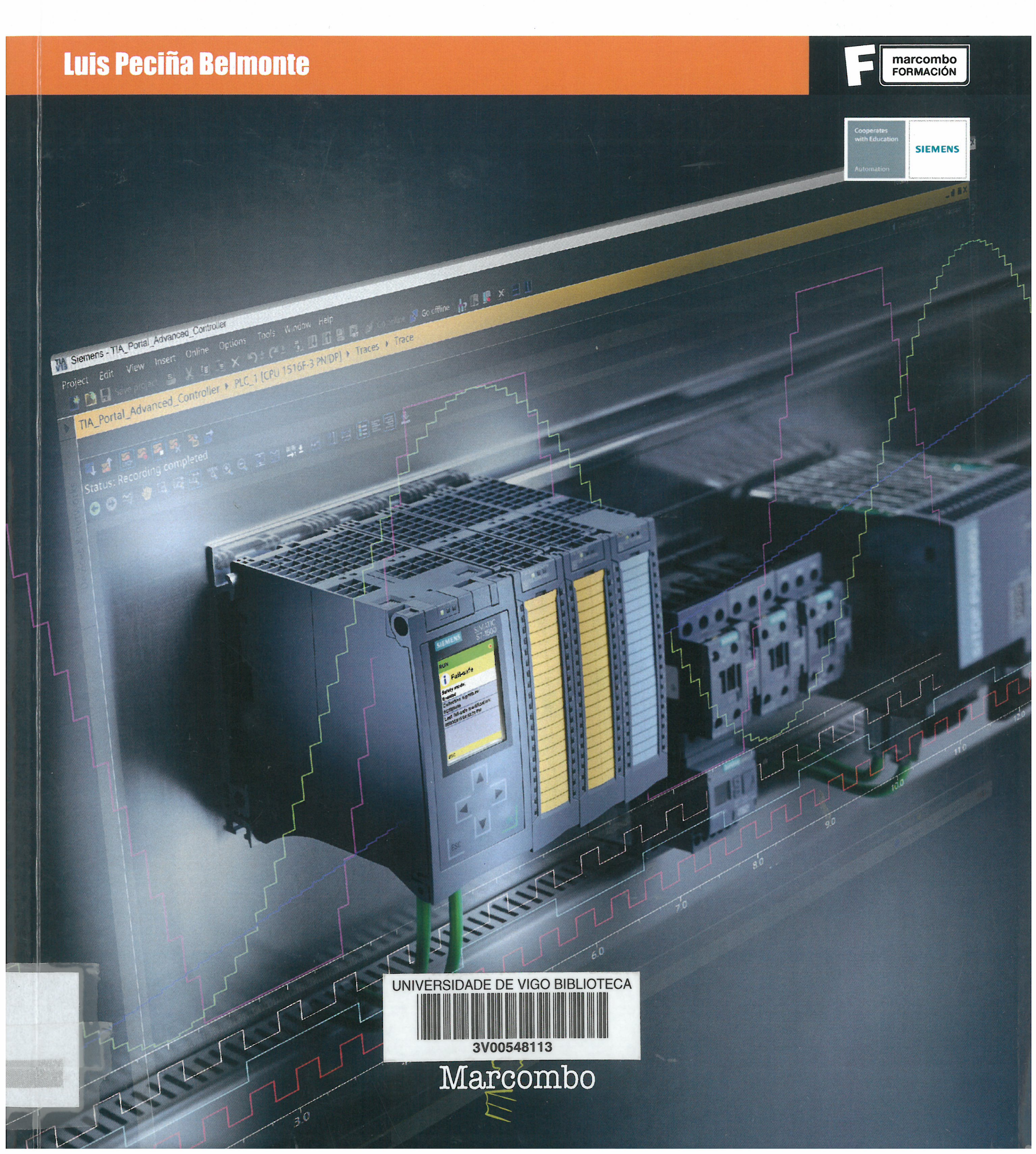
Programación de Autómatas Siemens S7 300 y S7 1500 AWL y SCL

Luis Peciña Belmonte

marcombo FORMACIÓN



Marcombo

PROGRAMACIÓN DE AUTÓMATAS SIEMENS S7-300 Y S7-1500 AWL Y SCL

Luis Peciña Belmonte



Programación de Autómatas Siemens S7-300 y S7-1500. AWL y SCL

Primera edición, 2017

Diseño de la cubierta: ENEDENÚ DISEÑO GRÁFICO

ISBN: 978-84-267-2459-5

D.L.: B-6930-2017

Printed in Spain

Prólogo

En 1958, Siemens Schuckert AG registró en las oficinas de patentes alemanas la marca SIMATIC, con la que sintetizaba los nombres de Siemens y Automatización en un solo concepto. Lo que se gestó como un sistema de control automático revolucionario se terminaría convirtiendo en el más innovador de los controladores PLC, al ofrecer soluciones que revolucionarían definitivamente el concepto de la automatización industrial.

En 1995, Siemens presentó al mercado la generación SIMATIC S7, que se convirtió en la generación más avanzada de autómatas Siemens y pasó a sustituir a la exitosa gama SIMATIC S5 de 1979.

SIMATIC S7, constituido inicialmente por los controladores S7-200, S7-300 y S7-400, ha ido evolucionando. En 2009 la gama fue ampliada con los controladores S7-1200 y en 2011 los SIMATIC S7-1500, claros sustitutos ambos de los controladores S7-200, S7-300 y S7-400, y programables con la plataforma Totally Integrated Automation Portal (TIA Portal).

TIA Portal ha sido una revolución. Por primera vez, un sistema único de control permite desde el desarrollo de la ingeniería y la puesta en marcha hasta la operación y monitorización de todos los accionamientos y componentes de la automatización: controladores y sistemas de supervisión.

En 2016, la nueva versión de TIA Portal (V14) da un salto cuántico, ya que a través de Addons permite la interconectividad del Totally Integrated Automation con las tecnologías disruptivas de la Industria 4.0. Estas son la computación en la nube, el Internet de las cosas (IoT), la movilidad, el *Big Data*, la analítica de datos, la impresión 3D o la robótica.

La era de la digitalización industrial supone todo un reto, tanto a nivel de medios como de conocimiento. Una investigación reciente del World Economic Forum sobre *The Future of Jobs* ha puesto de manifiesto la magnitud y la rapidez de estos cambios. Según este estudio, en 2020 más de la tercera parte de las competencias profesionales clave que se requieran todavía no son consideradas como cruciales en el mercado laboral actual.

Podemos afirmar que la Industria 4.0 requiere de una Educación 4.0 y, hoy más que nunca, Educación e Industria han de trabajar conjuntamente en la formación de los profesionales de ese futuro que ya es presente.

Por parte de Siemens, esta colaboración se enmarca dentro del proyecto **SCE —Siemens Automation Cooperates with Education—** en el que año tras año venimos desarrollando varias iniciativas orientadas a la formación de profesores y alumnos, entre las que destacan la organización de cursos especiales, jornadas técnicas, concurso de prototipo, patrocinio de las Olimpiadas de Formación Profesional *SpainSkills* y el Premio Nacional Don Bosco, organizado por el Colegio Salesianos de Zaragoza, así como aportación de documentación técnica y didáctica.

Nos congratula la iniciativa de la edición de este libro cuyo título es todo un acierto: ***Programación de autómatas Siemens S7-300 y S7-1500. AWL y SCL***, y de cuyo escritor, D. Luis Peciña, profesor del Centro de Formación Profesional Salesianos Zaragoza, puedo

destacar su alta profesionalidad técnica y docente avalada por más de 30 años dedicados al mundo de la formación.

FRANCISCO JAVIER CANO GATÓN

Responsable de Controladores SIMATIC y

Proyecto SCE – Siemens Cooperates with Education

División Factory Automation

Siemens, S.A.

Agradecimientos

Me gustaría hacer un pequeño homenaje a una persona que hizo que me dedicara a la enseñanza en la formación profesional. Mario Rubio García representa para muchos la viva imagen de la Formación Profesional, luchador incansable en su empeño de hacer de estas enseñanzas el mejor camino para una juventud en busca de una profesión de futuro. Mario no concebía la formación profesional sin los mejores profesionales y los mejores equipos. Su objetivo siempre era la superación, el liderazgo y la excelencia. Hizo de la Escuela Profesional Salesiana de Zaragoza uno de los mejores centros de España, reconocida por las empresas como una fábrica de buenos profesionales. Ideó el Premio Nacional Don Bosco, contactó con cientos de empresas, se relacionó con todos aquellos sectores que pudieran aportar algo para sus objetivos, siempre encaminados a ofrecer a los jóvenes la mejor formación técnica.

En el año 2015 nos dejó este gran hombre, que con su rigor y tesón hizo que, muchos de los profesores que pasamos por sus manos, nos convirtiéramos en profesionales cuyo objetivo siempre fuera la perfección y la rigurosidad en nuestra preparación para ofrecer lo mejor a nuestros alumnos.

Mario, los que te conocimos y trabajamos de cerca, seguimos tus directrices. Seguimos dando lo mejor de nosotros e intentamos estar a la vanguardia de la educación profesional. Tú siempre ibas por delante y marcabas el camino del futuro en la tecnología.

Y no puedo olvidarme en este apartado de la Congregación Salesiana, que me ha permitido poder ejercer unas de las profesiones más bonitas, la enseñanza, y más concretamente la enseñanza profesional. Ellos han puesto en mis manos la tecnología que luego intento transmitir a mis alumnos a base de entrega y dedicación.

Tampoco quiero olvidar a mis alumnos: los de ayer, los de hoy y los del mañana. Tengo una inmensa suerte de tener unos alumnos entregados con estas materias y así es muy fácil poder trabajar.

Por último, pero no menos importante, agradezco a mi familia y a mis amigos que soportan mis charlas de estos temas y de lo relacionado con la difícil labor de escribir un libro técnico.

Gracias a todos.

Índice

1. SISTEMAS DE NUMERACIÓN Y LÓGICA BINARIA

INTRODUCCIÓN

Un buen punto de partida en el estudio de los autómatas programables es comenzar por el principio, es evidente. Y ese principio es saber qué tipo de información (datos) puede recibir un autómata. Los autómatas programables, como cualquier otro sistema digital, solo pueden recibir unos o ceros. Es decir, que el sistema de numeración que se emplea para procesar información es binario. En este tema se va a estudiar cómo interpretar el sistema binario.

Igualmente es conveniente, pero no imprescindible, conocer la lógica binaria: los componentes básicos digitales y la forma de utilizarlos. En el libro hay alguno de los ejemplos que requiere conocer la forma de proceder con las funciones lógicas. Este tema pueden saltarlo todos aquellos que ya dispongan de esos conocimientos. Incluso para los que no deseen adquirir estos conceptos, no será un impedimento a la hora de asimilar la programación de los PLC.

SISTEMAS DE NUMERACIÓN: BINARIO Y HEXADECIMAL

Los sistemas de numeración siguen todos unos mismos criterios, una misma regla. Para poder llegar a entender el sistema binario, y cualquier otro, es necesario pararse a pensar en las reglas del sistema de numeración decimal, «nuestro sistema de numeración» por antonomasia.

Como se sabe, el sistema de numeración decimal tiene 10 dígitos, del 0 al 9. Cada sistema de numeración se caracteriza por su base, en el caso decimal la base es 10. Dentro de un número, la posición que ocupa cada dígito tiene un valor diferente, ese es el valor ponderal, es decir, el valor del dígito en función de la posición que ocupa. El siguiente ejemplo lo clarifica todo. El número 363 tiene dos dígitos iguales, pero no tienen el mismo valor dentro del número. Para saber el valor de cada dígito dentro del número es necesario elevar la base a la posición que ocupa dicho número, comenzando por la posición 0.

A continuación se indica el valor de cada dígito:

Posición 2	Posición 1	Posición 0
3	**6**	**3**
$10^2=100$	$10^1=10$	$10^0=1$

Se puede observar que la posición del número 3 colocado a la derecha tiene un valor ponderal de uno, el 6 tiene un valor ponderal de diez y el otro 3 un valor ponderal de cien. En el sistema decimal esos valores ponderales tienen un nombre: unidades, decenas y centenas. Eso no pasa en ningún otro sistema de numeración. La descomposición de un número se hace de la siguiente forma: **3 x 100 + 6 x 10 + 3 x 1 = 363**.

Y en este caso la descomposición da el número decimal correspondiente. Todos los sistemas de numeración siguen la misma regla.

Lo dicho anteriormente puede parecer obvio, pero resulta importante para poder estudiar el resto de los sistemas de numeración.

En el **sistema binario** se puede hacer un estudio paralelo al visto para el decimal. Lo único que cambia es la base. En este caso la base es 2 y, en consecuencia, los dígitos solo serán dos, el uno y el cero. A cada uno de estos dígitos (1,0) se le denomina bit. El bit se puede definir como la unidad de menor información binaria.

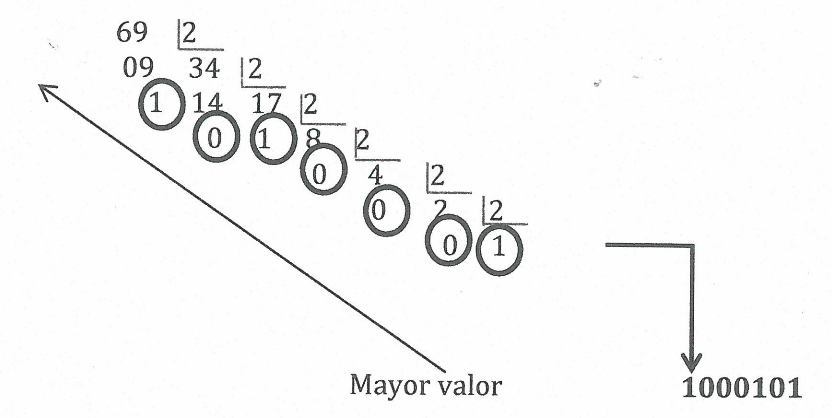
Si se realiza un estudio paralelo al decimal, obtenemos lo siguiente. Si el número binario es 1000101:

Posición 6	Posición 5	Posición 4	Posición 3	Posición 2	Posición 1	Posición 0
1	**0**	**0**	**0**	**1**	**0**	**1**
$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

La descomposición del número binario es:

$$\mathbf{1 \times 64 + 0 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 69}$$

En este caso el resultado de la descomposición es el número decimal que corresponde a dicho número binario. Hay que observar que los valores ponderales crecientes (hacia la izquierda) en el sistema binario se duplican con cada bit a la izquierda: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024... Teniendo esta idea clara, se puede hacer la conversión a decimal de una forma rápida. Igualmente se puede convertir de decimal a binario. Para este caso hay otro método que es ir dividiendo por dos, sin sacar decimales, hasta que el cociente sea menor que dos. El siguiente ejemplo está hecho según este criterio. El número 69 se va dividiendo por 2 hasta que el cociente sea menor de dos. Para encontrar el número binario se recogen todos los «unos» comenzando por el de mayor peso, que es el último cociente. Se continúa con los restos.



Siguiendo el paralelismo visto para el sistema decimal y binario, se puede estudiar el hexadecimal.

El **sistema de numeración hexadecimal** tiene 16 dígitos y su base es la 16. Los dígitos son 0......9 A B C D E F y su equivalencia en decimal desde el 0....9 coincide con el decimal, el A = 10, B = 11, C = 12, D = 13, E = 14 y F = 15. Este sistema se emplea para realizar, de forma más cómoda, el manejo de números binarios muy grandes.

Siguiendo la regla general, el número 3AF6 se puede descomponer en:

Posición 3	Posición 2	Posición 1	Posición 0
3	**A**	**F**	**6**
$16^3=4096$	$16^2=256$	$16^1=16$	$16^0=1$

$$3 \times 4096 + A \times 256 + F \times 16 + 6 \times 1$$

Sustituyendo los números hexadecimales por su valor decimal:

$$3 \times 4096 + 10 \times 256 + 15 \times 16 + 6 \times 1 = 15.094$$

En el siguiente ejemplo se puede observar la forma de trabajar. Sea el número binario 10111010110. Se deben coger grupos de 4 bits empezando por la derecha: 101 1101 0110. Ahora a cada grupo de cuatro bits se le calcula su valor en decimal y se convierte en su dígito correspondiente en hexadecimal.

101	**1101**	**0110**
5	**13**	**6**
5	**D**	**6**

Si lo que se desea es hacer la conversión contraria, de hexadecimal a binario, se hace del mismo modo. Cada dígito en hexadecimal se convierte a su correspondiente valor en binario, pero siempre cogiendo cuatro bits. Por ejemplo, el número F381 sería:

F	**3**	**8**	**1**
1111	**0011**	**1000**	**0001**

La tabla siguiente representa las equivalencias entre los sistemas decimal, binario y hexadecimal.

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B

DECIMAL	BINARIO	HEXADECIMAL
12	1100	C
13	1101	D
14	1110	E
15	1111	F

LÓGICA BINARIA

Tradicionalmente el control de procesos industriales automatizados se basaba en la utilización de **sistemas cableados,** constituidos por un conjunto de aparatos eléctricos (contactores, relés, temporizadores y elementos auxiliares). Estos elementos estaban unidos físicamente por medio de cables que formaban el circuito de mando.

Posteriormente la maquinaria industrial ha sido dotada de mayores medios y mecanismos, con los que el funcionamiento es menos dependiente del factor humano, más continuado y fiable. El desarrollo de la electrónica y de la informática ha hecho posible la automatización programable. Introducida en las cadenas de producción, ha transformado la industria en cuanto a la disminución de costes, variedad de productos que pueden ser fabricados utilizando una misma cadena, etc.

Algunas ventajas de la automatización programable son:

- ✓ Posibilidad de introducir modificaciones sin hacer cambios en la instalación, ya que el circuito de mando es un programa.
- ✓ Reducción del espacio ocupado por el conjunto de la instalación.
- ✓ Bajo coste de mantenimiento.
- ✓ Autolocalización de averías y posibilidad de controlar varios procesos con un mismo equipo.

LÓGICA DE UN AUTOMATISMO

Actualmente la mayoría de los procesos industriales y máquinas están automatizados. Ascensores, semáforos, calefacciones, etc., son algunos ejemplos de las muchas aplicaciones cuyo funcionamiento está controlado por automatismos. Hoy nuestro entorno industrial y doméstico está altamente automatizado y el futuro es realmente prometedor, ya que estamos inmersos en la cuarta revolución industrial, donde el Internet de las cosas (IoT) y, más concretamente el Internet Industrial de las cosas (IIoT), supone la clave para lograr que, en un futuro muy cercano, se tenga control total sobre todos los procesos industriales.

Pero para llegar a esos niveles de tecnificación es necesario tener muy claro los fundamentos básicos, comenzando por el sistema binario y el álgebra de Boole. Esos conceptos se aplican al diseño de automatismos, con independencia de la tecnología utilizada en su implementación, y son el soporte de la lógica programable aplicada a los autómatas programables.

El autómata es un sistema digital y como tal solo entiende de «unos» y «ceros». Ya se ha tratado el sistema binario, ahora se deberá aplicar y utilizar. En esta parte se van a estudiar los sistemas digitales electrónicos básicos. Servirá para conocer cómo trabaja un autómata y también para realizar los primeros programas en el autómata.

Los valores digitales sabemos que son uno o cero, y cada uno de ellos es un bit. En la vida real esos dos valores se pueden representar como: tensión disponible o tensión no disponible, contacto abierto o contacto cerrado, circula corriente o no circula, etc.

BIT, BYTE, PALABRA, DOBLE PALABRA

El **bit** es la unidad de una señal binaria. Un bit es la unidad de menor información binaria y puede adoptar los estados «1» y «0».

Un **byte (B)** está formado por 8 bits.

La **palabra binaria (W)** es el número de bits que una máquina digital puede procesar de una sola vez. En muchos PLC la palabra binaria es de 16 bits, es decir dos bytes. En los autómatas que se estudiarán en este libro, la longitud de la palabra binaria es de 16 bits.

La **doble palabra (D)** está formada por dos palabras binarias, es decir, cuatro bytes o, lo que es lo mismo, 32 bits.

CÓDIGO BCD

Hay que diferenciar lo que es un código binario y un número binario. El número binario sigue la regla para todos los sistemas de numeración. El código es un «acuerdo» que todos aquellos que lo usan deben conocer para descifrarlo.

Un código binario importante es el BCD (decimal codificado en binario). Sirve para facilitar la introducción de datos numéricos decimales a las máquinas digitales. Su conversión es directa y fácil.

En un número binario codificado en BCD se mantiene el valor de la posición de los números decimales (potencias de base 10), aunque los dígitos del número decimal se representan en binario.

Por ejemplo, el número decimal 359 en BCD es 0011 0101 1001. Cada dígito decimal se convierte a su correspondiente en binario.

0011	**0101**	**1001**
3	**5**	**9**

En un número BCD no puede haber ningún valor binario que supere el 9 en decimal, ya que **en BCD solo se usan los números del 0 al 9.**

FUNCIONES LÓGICAS Y PUERTAS DIGITALES

Toda la tecnología actual tiene como base componentes electrónicos que llevan integradas miles de puertas lógicas: ordenadores, autómatas, tabletas, móviles, etc. Las puertas lógicas básicas son AND, OR, NAND y NOR. Agrupando y formando circuitos lógicos con este tipo de puertas se puede construir un automatismo. Como ya se ha dicho anteriormente, se va a realizar el estudio de este tipo de puertas como base de partida de los primeros programas con el autómata.

Para realizar el estudio de cualquier circuito lógico es necesario establecer su **tabla de la verdad**. Atendiendo a dicha tabla se podrá saber el comportamiento del circuito. Además de la tabla de

verdad, todo circuito lógico tendrá su **función lógica**, que es la representación matemática del comportamiento del circuito. Es decir que, tanto la tabla de verdad como la función, nos relacionan el comportamiento de las salidas dependiendo de cómo estén las entradas en el circuito.

En la tabla siguiente se estudian las puertas lógicas básicas indicando su símbolo, su tabla de verdad, su función y su equivalencia eléctrica, si la tienen. Todas las puertas que se estudian son de dos entradas.

	AND (Y)	OR (O)	NAND (NO Y)	NOR (NO O)
Símbolo	a 1, b 2, 3 S	a 1, b 2, 3 S	a 1, b 2, 3 S	a 2, b 3, 1 S
Función	S = a · b	S = a + b	$S = \overline{a \cdot b}$	$S = \overline{a + b}$
Tabla de verdad	a b S 0 0 0 0 1 0 1 0 0 1 1 1	a b S 0 0 0 0 1 1 1 0 1 1 1 1	a b S 0 0 1 0 1 1 1 0 1 1 1 0	a b S 0 0 1 0 1 0 1 0 0 1 1 0
Equivalencia	a b S	a b S	b CONTACTO CERRADO b CONTACTO ABIERTO	

Existen otras puertas lógicas como la función NOT, también llamada negación. La función NOT da como resultado el inverso del estado de la variable de entrada. Si la variable de entrada vale 1, en la salida de una NOT hay 0, y viceversa. El símbolo electrónico es:

a 1 2 S

Su función es: $S = \overline{a}$.

Formas de representar la negación		
En la función se coloca una raya encima de la variable o variables	En el circuito electrónico se añade un circulito	En el eléctrico se coloca una raya cruzada en el contacto
$S = \overline{a . b}$	a 1, b 2, 3 S	b

Hay dos puertas lógicas más que tienen una menor aplicación. Son la OR Exclusiva (XOR) y la NOR Exclusiva (XNOR). A continuación se indican en la tabla su símbolo y función:

	XOR	XNOR
Símbolo	a, b → S	a 1, b 2 → 3 S
	XOR	**XNOR**
Función	$S = a \oplus b$	$S = \overline{a \oplus b}$
Tabla de verdad	a b S: 0 0 0 / 0 1 1 / 1 0 1 / 1 1 0	a b S: 0 0 1 / 0 1 0 / 1 0 0 / 1 1 1

SISTEMAS COMBINACIONALES

Los circuitos vistos hasta ahora son las puertas básicas con las que se realizan circuitos más complejos. En las aplicaciones del autómata no nos van a interesar los circuitos electrónicos, pero sí las aplicaciones que se pueden realizar con sus funciones para, posteriormente, convertirlas en un programa que introduciremos en el PLC. Un circuito combinacional es el que está formado por diferentes puertas que tienen varias entradas y una o más salidas. Las salidas responden a una combinación de las entradas.

Vamos a ver un ejemplo de un circuito combinacional de 3 entradas A B C y una sola salida S. Para saber el número de combinaciones diferentes que habrá con un número determinado de entradas basta con elevar 2 al número de entradas. Para este caso de tres entradas, la tabla de verdad tiene $2^3 = 8$ combinaciones diferentes.

N.º decimal	c	b	a	S
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

En la columna n.º decimal se puede ver el equivalente decimal al código binario. Las entradas, **a b c**, asumirán las combinaciones desde 0 hasta 7, es decir, 8 combinaciones. La salida indica el comportamiento de esa función. En este caso la salida nos dará un 1 lógico solo en las combinaciones 0, 2 y 3. En el resto de combinaciones la salida vale 0.

Las entradas a, b, c, pueden ser pulsadores, interruptores, finales de carrera, sensores, etc. La salida o salidas pueden ser diodos led, relés, electroválvulas, etc.

Para resolver este tipo de circuito con puertas comerciales, hay que escribir la **función de la salida.** Para ello debemos tener en cuenta los unos de la salida S.

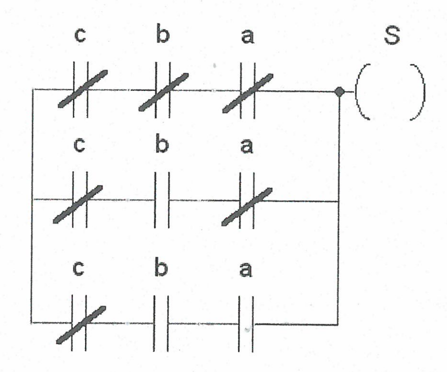
El valor 0 de una entrada le corresponde a la variable de entrada negada. El valor 1 corresponde a la variable de entrada sin negar. **Una entrada negada es un 0, y se puede representar con un interruptor cerrado en reposo, mientras que una entrada no negada es un 1, y se puede representar con un interruptor abierto en reposo.**

La función canónica de la tabla de verdad es:

$$S=(\overline{A}\cdot\overline{B}\cdot\overline{C})+(\overline{A}\cdot B\cdot\overline{C})+(A\cdot B\cdot\overline{C})$$

Cuando, como en este caso, se cogen solo los «unos», se dice que la función se representa con los términos **minterm**, y en esta forma el resultado es una función **con sumas de productos**, lo que quiere decir que debe cumplirse cualquiera de esos tres productos para que la salida sea UNO.

Si tuviéramos que realizar el circuito de la tabla de verdad con interruptores, llevaríamos a cabo este esquema.



La salida se activa cuando se cumple, al menos, alguno de estos tres casos:

- ✓ las tres entradas están negadas (sin pulsar);
- ✓ A y C sin pulsar, y pulsamos B;
- ✓ C sin pulsar, A y B pulsados.

En los demás casos la salida no se activa.

Ejercicio:

A partir de la tabla de verdad siguiente, escribir la función con minterm usando por tanto los estados altos de las salidas.

N.º decimal	c	b	a	S	S1
0	0	0	0	0	1
1	0	0	1	1	0

2	0	1	0	0	1
3	0	1	1	0	1
4	1	0	0	0	0
5	1	0	1	0	0
6	1	1	0	1	0
7	1	1	1	0	0

Las funciones de las dos salidas serán:



$$S = (A \bullet \overline{B} \bullet \overline{C}) + (\overline{A} \bullet B \bullet C)$$

$$S1 = (\overline{A} \bullet \overline{B} \bullet \overline{C}) + (\overline{A} \bullet B \bullet \overline{C}) + (A \bullet B \bullet \overline{C})$$

La función S1 coincide con la del circuito anterior porque no ha variado.

Como ejercicio se debe dibujar el esquema de contactos del ejemplo anterior.

TÉCNICAS DE SIMPLIFICACIÓN DE FUNCIONES LÓGICAS CON EL MÉTODO GRÁFICO DE LOS MAPAS DE KARNAUGH

Cuando la función o funciones resultantes de la combinación de las salidas son largas, conviene simplificar términos para disminuir el número de contactos. Si el circuito se tiene que realizar con un autómata programable o con contactores, esto se traduce en un ahorro de instrucciones o de contactores.

El método que se va a utilizar para simplificar es el de los mapas de Karnaugh, ideado en 1953 por Maurice Karnaugh.

Este sistema de simplificación consiste en realizar unas tablas (mapas) con tantos cuadros o celdas como combinaciones tiene la tabla de verdad. Por ejemplo, 4 variables de entrada:

2^4 = 16 celdas.

Generalmente, en los mapas se colocan las variables de mayor peso en vertical y las de menor peso en horizontal.

Ejemplos de mapas de 3 y 4 variables de entrada

Para tres variables:

A \ BC	00	10	11	01
0				
1				

8 combinaciones de salida

Para cuatro variables:

AB \ CD	00	10	11	01
00				
10				
11				
01				

16 combinaciones de salida

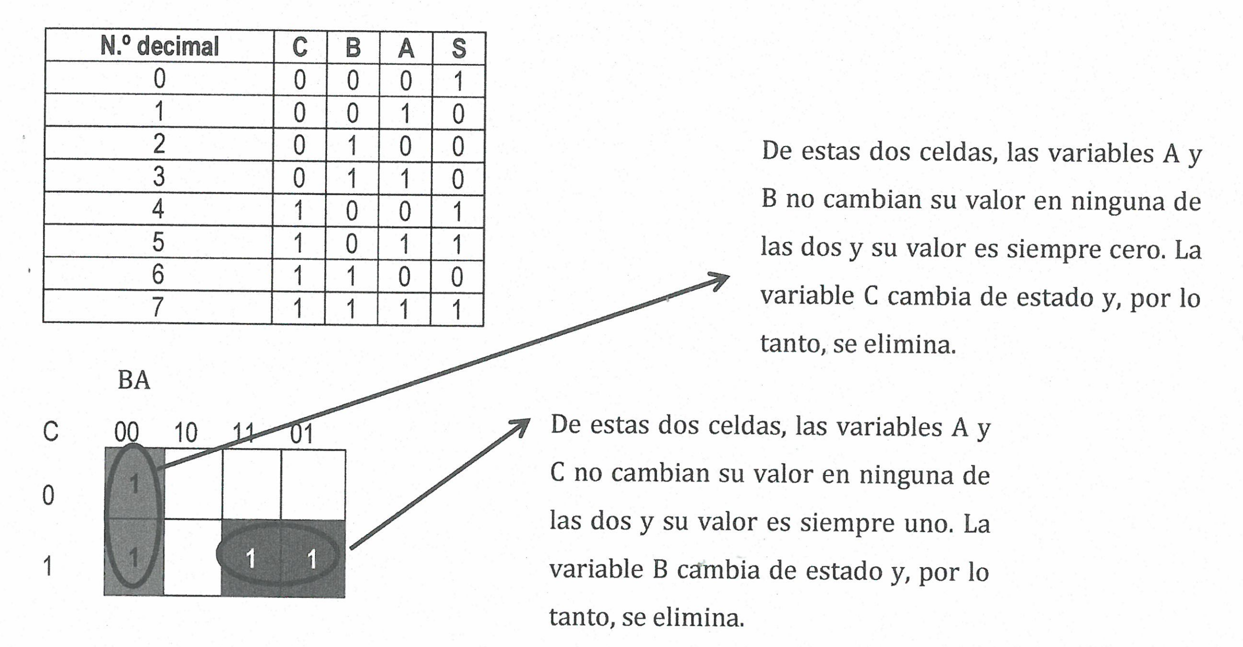
Una vez realizado el mapa, el siguiente paso es colocar los unos (minterm) en los cuadros correspondientes **a las combinaciones de entradas, que activan las salidas.**

Cuando los cuadros están completos, pasamos a enlazar los unos en grupos que sean potencia de 2 (2, 4, 8, 16, etc.). No debe quedar ningún uno sin agrupar y puede haber celdas comodín, es decir, que sirvan para diferentes enlaces.

Los agrupamientos se hacen con celdas que tienen unos adyacentes, en sentido derecha, izquierda, abajo, arriba, nunca en diagonal. La primera y última fila, así como la primera y última columna, se consideran adyacentes.

Podemos ahora simplificar, **eliminando la variable o variables, que cambian de valor dentro de la agrupación.** Si algún uno no puede agruparse con nadie, se queda la combinación entera, sin simplificar ninguna variable.

Ejemplo:



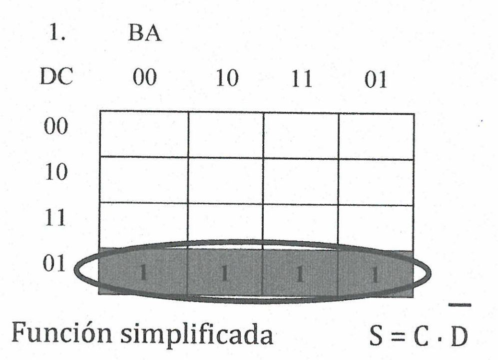
N.º decimal	C	B	A	S
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Función simplificada:

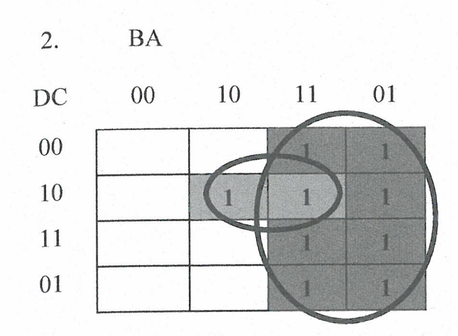
$$S = A \cdot C + \overline{A} \cdot \overline{B}$$

Más ejemplos:

Dados los siguientes diagramas de Karnaugh, resolver la simplificación.



Función simplificada $S = C \cdot \overline{D}$



Función simplificada: $S = A + B \cdot \overline{C} \cdot D$

2. TIPO DE DATOS Y VARIABLES

INTRODUCCIÓN

La memoria del autómata se distribuye de la siguiente forma:

- Memoria de programa: es la zona donde se guarda el programa una vez compilado. Se ejecuta secuencialmente.
- Imagen de entradas y salidas: son la PAE y la PAA, es decir, el área donde se guardan las imágenes de las entradas y salidas físicas.
- Marcas de memoria: es la zona dedicada a las marcas, que son los registros donde guardaremos los datos intermedios. Podíamos decir que son los «relés auxiliares» en los sistemas cableados.
- E/S de periferia: la zona reservada para el direccionamiento de las entradas y salidas reales.
- Zona para temporizadores y contadores: son los registros dedicados a guardar los valores de los temporizadores y contadores.
- Zona para datos temporales: como las pilas de los saltos y otros datos que se usan durante la programación.
- Zona para acumuladores: los acumuladores son registros de 32 bits que se utilizan para ciertas operaciones importantes. Los autómatas de la serie 300 disponen de dos acumuladores y los de la serie 400, de cuatro.

La capacidad de cada una de estas zonas depende de cada autómata. En el capítulo 3 se pueden observar diferentes tipos de PLC y sus características.

TIPOS DE MÓDULOS

Los autómatas de la serie 300 y 400 disponen de unos grupos de programación que van a servir para realizar los programas mejor organizados, más estructurados y modulares.

- Bloque de organización (OB): son bloques donde se escribe programa. Hay tres tipos:
  - OB1: es el bloque donde se carga el programa principal escrito por el usuario. Se ejecuta cíclicamente.
  - OB de error: hay variados bloques de este tipo y en él se escribe el programa que deseamos que se ejecute cuando se produzca un error concreto.
  - OB de arranque: en este módulo se programa todo lo que se desea hacer cuando el programa arranque (OB100). Solo se ejecuta cuando el PLC pasa de STOP a RUN.

- Bloque de función (FC): son módulos de programa que se ejecutan solo cuando se desea. Se llaman desde otros bloques, por ejemplo, desde el OB1 u otra función.
- Boque de función con datos (FB): son similares a las funciones. La diferencia es que los FB disponen de una zona de memoria (DB) para guardar datos.
- Bloque de datos (DB): son zonas de memoria donde se guardan datos. Disponen de un tratamiento especial.
- Módulos de funciones especiales (SFB): son bloques de programa ya realizados y que llevan a cabo acciones complejas, por ejemplo, un PID.
- Módulos de funciones del sistema (SFC): son funciones integradas en la CPU y que el usuario puede utilizar.

TIPOS DE DATOS

Los operandos de las instrucciones se componen de un dato que puede ser de distintos tipos. Estos tipos pueden ser:

E	entrada	T	temporizador
A	salida	Z	contador
M	marca	DB	módulo de datos
P	periferia		

Cada uno de estos tipos se puede direccionar de cuatro formas diferentes:

X: bit (1 bit)
B: byte (8 bits)
W: palabra (16 bits)
D: doble palabra (32 bits)

Los contadores y temporizadores utilizan solo palabras (W).

MARCAS DE MEMORIA

Las marcas de memoria son zonas de la memoria que pueden utilizarse de forma temporal y auxiliar. Se dispone de 256 marcas de 1 byte cada una, aunque esto varía dependiendo de la CPU.

Pueden ser:

- Marcas de bit: si queremos utilizar tan solo un bit. La forma de direccionar dichas marcas será indicando el byte y el bit. Por ejemplo, M3.6 indica que estamos utilizando el bit 6 del byte o marca 3.
- Marcas de byte: en este caso utilizamos el byte completo. Se indicará de esta forma: MB4.
- Marcas de palabra: utilizamos dos bytes y se indican como MW4; en este caso estamos utilizando el byte B4 y el B5.

B4 B5

5.7	5.6	5.5	5.4	5.3	5.2	5.1	5.0	4.7	4.6	4.5	4.4	4.3	4.2	4.1	4.0

W5

- Marcas de doble palabra: se utilizan cuatro bytes. Por ejemplo, MD4 será el formato de la doble palabra de marca 4, formada por los bytes 4, 5, 6 y 7.

B4 B5 B6 B7

4.7	4.6	4.5	4.4	4.3	4.2	4.1	4.0	5.7	5.6	5.5	5.4	5.3	5.2	5.1	5.0	6.7	6.6	6.5	6.4	6.3	6.2	6.1	6.0	7.7	7.6	7.5	7.4	7.3	7.2	7.1	7.0

W4 W6

D4

ENTRADAS Y SALIDAS

Las entradas y salidas, cuando se utilizan en el programa, son accesibles desde la memoria interna del autómata, en las zonas PAE y PAA, respectivamente. La capacidad máxima de E/S direccionables depende del PLC utilizado. Como máximo podemos direccional 65536 bytes para cada tipo de E/S.

La forma de direccionar las E/S es igual que para las marcas, tan solo hay que cambiar la letra E para la entrada y la A para la salida.

Estos serían unos ejemplos:

E 1.5	Entrada de bit (byte 1, bit 5)	A 1.5	Salida de bit (byte 1, bit 5)
EB 1	Entrada de byte 1	AB 1	Salida de byte 1
EW 1	Entrada de palabra 1 (byte 1 y 2)	AW 1	Salida de palabra 1 (byte 1 y 2)
ED 1	Entrada de doble palabra 1 (byte 1, 2, 3 y 4)	AD 1	Salida de doble palabra 1 (byte 1, 2, 3 y 4)

Las entradas y salidas pueden ser digitales o analógicas.

Si queremos acceder de forma directa a la periferia, es decir a la E/S directamente sin pasar por sus imágenes en memoria (PAE y PAA), debemos anteponer la letra P. Por ejemplo:

PEB 1	Entrada de periferia, byte 1	PAB 1	Salida de periferia, byte 1
PEW 1	Entrada de palabra de periferia 1 (byte 1 y 2)	PAW 1	Salida de palabra de periferia 1 (byte 1 y 2)
PED 1	Entrada de doble palabra de periferia 1 (byte 1, 2, 3 y 4)	PAD 1	Salida de doble palabra de periferia 1 (byte 1, 2, 3 y 4)

No se puede acceder a periferia a nivel de bit.

REGISTROS

ACUMULADOR

El acumulador es un registro de 32 bits (4 bytes). Se emplea para manejar bytes, palabras o dobles palabras. Hay muchas instrucciones que utilizan el acumulador y según se estudien iremos viendo cual es la función que desempeña en cada caso. El S7-300 tiene dos acumuladores (ACU 1 y ACU 2) y el S7-400 cuatro (ACU1, ACU 2, ACU 3 y ACU 4).

S7-300:

ACU 1 (H)	ACU1 (L)	ACU2 (H)	ACU2 (L)

S7-400:

ACU1 (H)	ACU1 (L)	ACU2 (H)	ACU2 (L)	ACU3 (H)	ACU3 (L)	ACU4 (H)	ACU4 (L)

(H) = 16 bits bajos de cada acumulador

(L) = 16 bits altos de cada acumulador

REGISTRO DE ESTADO

Es un registro de 16 bits que da información sobre la situación del PLC después de ciertas operaciones. Del registro o palabra de estado solo se utilizan 9 bits.

La estructura de ese registro es la siguiente:

x	x	x	x	x	x	x	RB	A1	A0	OS	OV	OR	STA	RLO	ER

- BIT 0 (ER): 0 indica que la siguiente línea se ejecuta como nueva consulta (inhibida). En este estado la consulta se almacena directamente en RLO (ver 4.1).
- BIT 1 (RLO): resultado lógico. Aquí se realizan las operaciones a nivel de bit (AND, OR, etc.).
- BIT 2 (STA): bit de estado. Solo sirve en el test de programa.
- BIT 3 (OR): se requiere para el proceso Y delante de O. Este bit indica que una operación Y ha dado valor 1 y en las restantes operaciones es 0.
- BIT 4 (OV): bit de desbordamiento. Se activa (1) por una operación aritmética o de comparación de coma flotante tras producirse un error (desbordamiento, operación no admisible o relación incorrecta).

- BIT 5 (OS): bit de desbordamiento memorizado. Se activa junto con OV e indica que previamente se ha producido un error. Solo puede cambiar a cero con la instrucción SPS, una operación de llamada a módulo, o porque se ha alcanzado el fin del módulo.
- BITS 6 (A0) y 7 (A1): códigos de condición. Dan información sobre los resultados o bits siguientes:

  - resultado de una operación aritmética,

  - resultado de una comparación,

  - resultado de una operación digital,

  - bits desplazados por una instrucción de desplazamiento o rotación.

- BIT 8 (RB): resultado binario. Permite interpretar el resultado de una operación de palabras como resultado binario e integrarlo en la cadena de combinaciones lógicas binarias.

DATOS SIMPLES

Los datos simples son los datos de tipo booleano, byte, palabra, doble palabra (enteros y reales), tiempos, de fecha y de tipo carácter. Cumplen la norma IEC-1131-3.

Para poder representarlos en un programa se debe conocer su notación. En la siguiente tabla podemos comprobar cómo hacerlo.

TIPO	N.º BITS	FORMATOS	RANGO	EJEMPLO
BOOL	1	Texto/número	1/0 TRUE/FALSE	1 True
BYTE	8	Hexadecimal	De B#16#0 a B#16#FF	L B#16#F2
WORD	16	Binario	De 2#0 a 2#1111_1111_1111_1111	L 2#1001-_0000_0101_0000
		Hexadecimal	De W#16#0 a W#16#FFFF	L W#A3FF
		BCD	De C#0 a C#999	L C#125
		Entero sin signo	De B#(0,0) a B#(255,255)	L B#(23,45)
DWORD	32	Binario	De 2#0 a 2#1111_1111_111_1111_1111_1111_111 1_1111	L 2#1000_0011_1100_0001_ 1000_1100_1100_1111_01
		Hexadecimal	De DW#16#0000_0000 a DW#16#FFFF_FFFF	L DW#16#FE34_FFA3
		Entero sin signo	De B#(0,0,0,0) a B#(255,255,255,255)	L B#(34,28,134,67)

TIPO	N.º BITS	FORMATOS	RANGO	EJEMPLO
INT	16	Entero con signo	De -32768 a 32767	L 1
DINT	32	Entero con signo	De L#-2147483648 a L#2147483647	L L#1
REAL	32	Número en coma flotante	De ±3,402823e^{+38} a ±1175495e^{-38}	L 1,458e^{+8}
S5TIME	16	Tiempo Simatic (10 ms)	De S5T#0h0m0s10ms a S5T#2h46m30s0ms	L S5T#1h4m
TIME	32	Tiempo IEC (1 ms)	De -T#24d20h31m23s648ms a T#24d20h31m23s647ms	L T#od2d0h24m
DATE	16	Fecha IEC (1 día)	De D·1990-1-1 a D#2168-12-31	L D#2006-5-10
TIME_OF_DAY	32	Hora del día (1 ms)	De TOD#0:0:0 a TOD#23:59:59,999	L TOD#2:09:4
Carácter	8	ASCII	'A', 'b'.....	L 'G'

3. AUTÓMATA: S7-300 y S7-1500

INTRODUCCIÓN

Un autómata programable es un equipo electrónico de control con un cableado interno independiente del proceso a controlar, que se adapta a dicho proceso mediante un programa específico que contiene la secuencia de operaciones a realizar y hace uso de sus entradas y salidas.

Las señales de entrada pueden proceder de:

- Elementos digitales: finales de carrera, pulsadores o detectores.
- Elementos analógicos: sensores de temperatura, dispositivos de salida en tensión o corriente continuas.

Las señales de salida son órdenes digitales, todo o nada, o señales analógicas de tensión o corriente, que se envían a los elementos indicadores y actuadores del proceso, como lámparas, contactores, válvulas, etc.

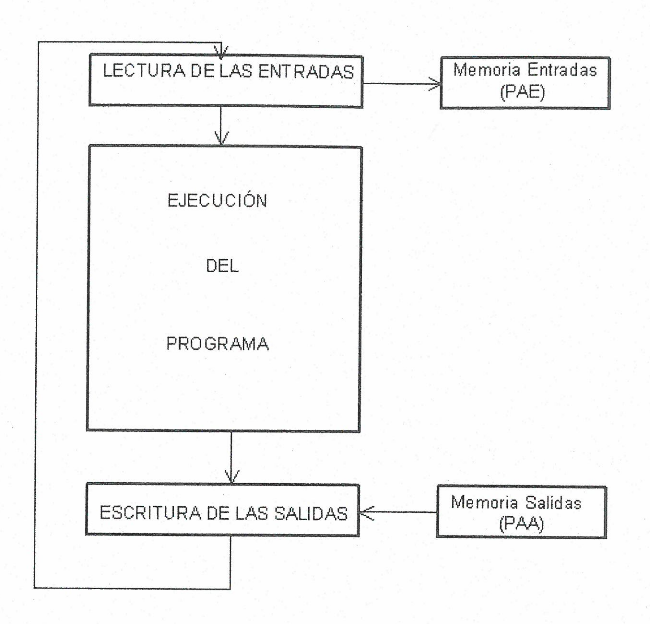
El autómata se encarga de la parte de control del automatismo. Por lo tanto, en un sistema cableado la parte de mando es sustituida por el autómata, nunca la parte de potencia.

TIEMPO DE CICLO

El autómata ejecuta su programa de una manera secuencial. Se llama tiempo de ciclo (TC) al tiempo que tarda el autómata en realizar un ciclo completo, es decir, desde que LEE las entradas hasta que ESCRIBE las salidas.

Es muy importante conocer cómo trabaja un autómata. Es un principio básico, sencillo, que si se desconoce, puede ocasionar muchos problemas. Como norma general, cuando se arranca el autómata, hace una lectura del estado de todas las entradas digitales y la guarda en la memoria interna del PLC. A continuación, el autómata ejecuta de forma secuencial cada una de las órdenes que hay en el programa principal (OB1). Dentro de ese programa habrá lecturas de las entradas. Esas lecturas se realizan de la memoria de entradas (PAE), no de la entrada física real. También durante la ejecución del programa habrá activaciones de salidas. Está activación de las salidas no activa la salida física real en el instante de la ejecución de la orden. Se actualiza en la memoria de las salidas (PAA). Una vez terminado el programa, el PLC vuelca la memoria de salidas sobre las salidas físicas y las actualiza en ese momento. Ahora el ciclo se repite.

Estos tiempos de ciclo varían dependiendo del tipo de CPU y, lógicamente, estos tiempos también dependen de la cantidad de instrucciones del programa. Es peligroso aproximarse al tiempo máximo, ya que, mientras se está ejecutando el programa, no se leen las entradas ni se activan las salidas. El *watchdog* (perro vigía) es el encargado de vigilar este ciclo, de forma que, si se pasa, lleva el autómata a STOP.



PANORAMA ACTUAL S7-300/1200/1500

Los autómatas S7-300, S7-1200 y S7-1500 son una marca registrada de SIEMENS. La evolución de los diferentes autómatas de la marca Siemens ha ido a la par con el incremento del uso de las tecnologías de la información y de las redes de comunicaciones. Hoy se requieren características diferentes a las exigidas hace 20 años. A nivel industrial se dispone de las series 300/1200/1500. Cada serie se utiliza para aplicaciones y necesidades concretas. En la Figura 1 se puede apreciar el posicionamiento de los diferentes PLC de Siemens antes de la aparición del S7-1200.

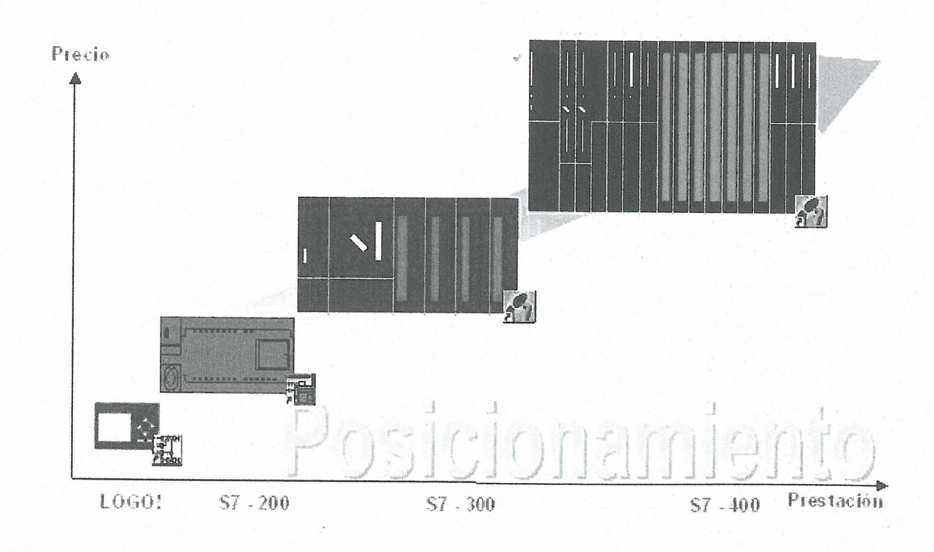


Figura 1

El PLC LOGO es un programador utilizado para aplicaciones de tipo domótico. El S7-200 es un PLC que se dejó de fabricar hace algunos años. Dicho autómata fue sustituido por el S7-1200,

destinado a aplicaciones industriales de control de pequeño tamaño o automatización de máquinas.

Las series S7-300 y S7-400 son autómatas utilizados para grandes aplicaciones industriales de control. Hoy conviven junto al nuevo PLC S7-1500. Es seguro que el S7-1500 será el que, en un tiempo no muy lejano, sustituirá al S7-300/400 En la nueva concepción de Siemens cabe destacar que todos los PLC nuevos pueden ser programados en el mismo lenguaje que el S7-300.

Con la aparición del S7-1200, Siemens sacó una plataforma de programación nueva denominada TIA PORTAL.

En este libro se va a trabajar con los autómatas de la serie S7-300 y S7-1500. Aunque el PLC S7-1500 ha llegado para sustituir al S7-300, es conveniente tener un profundo conocimiento del S7-300 por dos motivos. Uno, porque la realidad y la experiencia nos demuestran que, desde el momento que se descataloguen los S7-300, en las industrias habrá una gran cantidad de este tipo de autómatas. El otro motivo es que la programación del S7-1500 es igual que la del S7-300, aunque el S7-1500 dispone de un mejor tratamiento de las instrucciones, sobre todo en la norma de estandarización IEC 61131. Respecto al *software*, se puede decir lo mismo, es necesario conocer el STEP 7 y, por supuesto, el nuevo entorno de programación TIA PORTAL.

En la Figura 2 se muestra la clasificación de los autómatas de Siemens hoy en día.

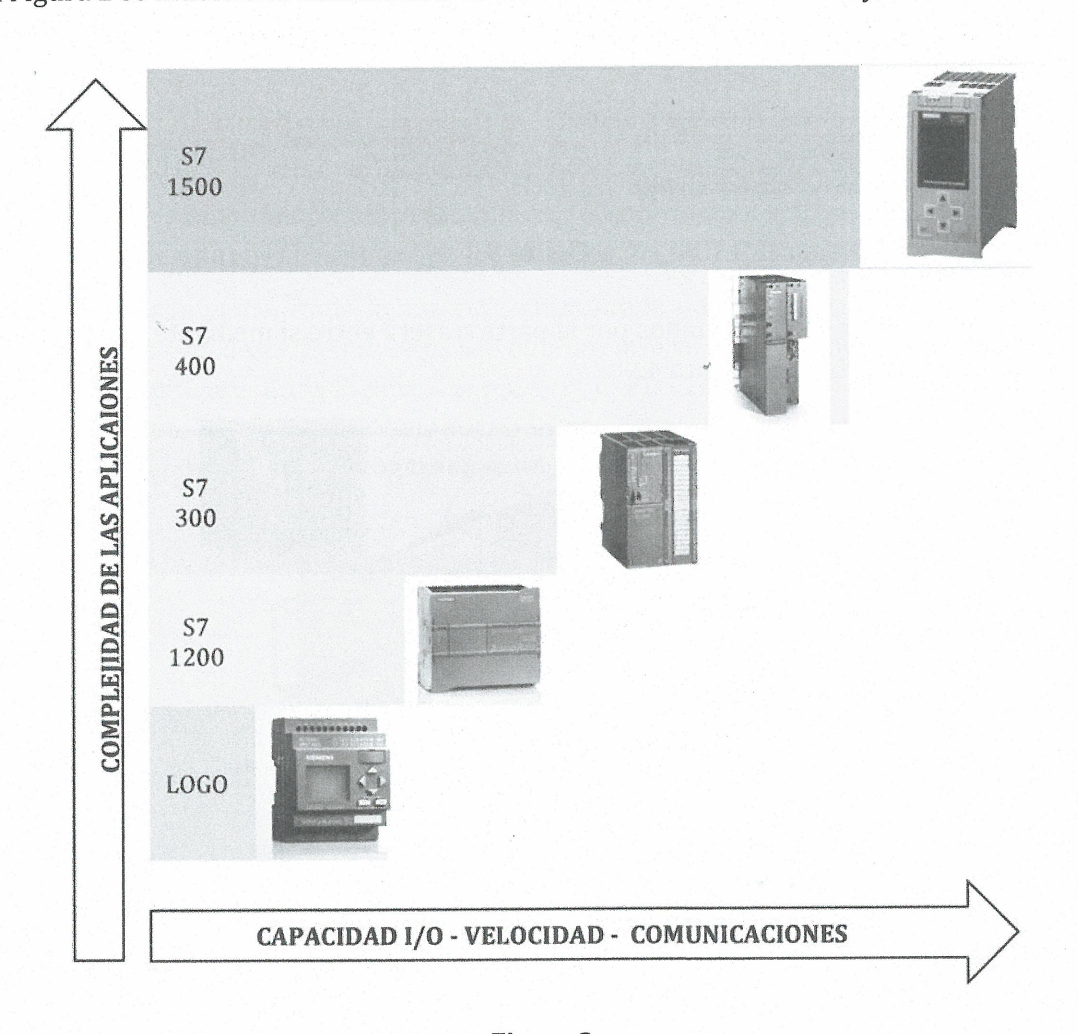


Figura 2

Existen muchos tipos diferentes de autómatas dentro de cada una de las series. Las diferencias son de memoria, número de temporizadores y contadores, de comunicaciones, etc.

AUTÓMATA S7-300

Todo autómata está constituido como mínimo por la fuente de alimentación, la CPU y los módulos de entrada y salida. Podrá disponer de otros módulos específicos, como módulos de comunicaciones, módulos de contador rápido, etc. En la Figura 3 se aprecian estos elementos en un PLC S7-300.

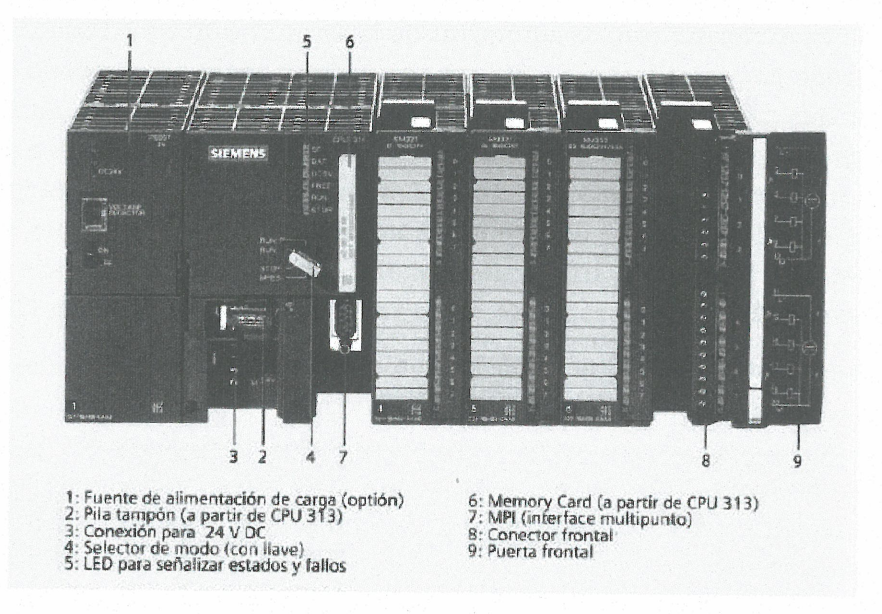


Figura 3

Los diferentes módulos van conectados por la parte trasera entre sí mediante un bus en forma de U, tal como se aprecia en la Figura 4.

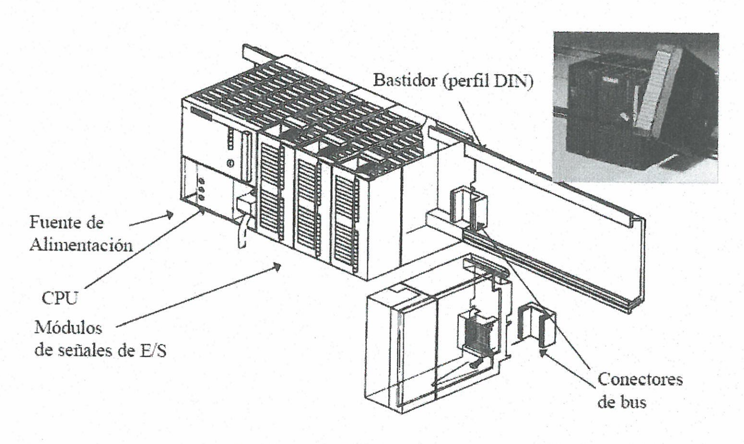


Figura 4

Hay autómatas compactos y modulares. Los que tienen la CPU compacta disponen internamente de un determinado número de E/S y se les pueden añadir otras externamente. Estas CPU llevan la letra C en su referencia. Estos PLC pueden ser operativos sin añadir más módulos. En la serie S7-300, y también en la S7-1500, es necesario añadirles una fuente de

alimentación. Los S7-1200 llevan también integrada la fuente. Los modulares son aquellos que no son operativos si no se incluyen módulos de entrada/salida. En las tablas siguientes se pueden apreciar algunas características de los PLC de la serie 300.

	CPU 315-2PN/DP	CPU 317-2PN/DP
Memoria de trabajo	128 KB	256 KB
Tarjeta de memoria (MMC)	SI	SI
Cantidad de DB	1-16000 (64 KB)	1-16000 (64 KB)
Cantidad de FB	0-7999 (64 KB)	0-7999 (64 KB)
Cantidad de FC	0-7999 (64 KB)	0-7999 (64 KB)
Memoria OB	64 KB	64 KB
Tiempo CPU BIT (min)	0,06 μs	-
Tiempo CPU WORD (min)	0,09 μs	0,03 μs
N.º de contadores	256	512
Rango Contadores	0-999	0-999
Contadores IEC	SI	Si
N.º temporizadores	256	512
Rango temporizadores	10 ms a 9990 s	10 ms a 9990 s
Temporizadores	SI (SFB)	SI (SFB)
Web server	SI (Solo lectura)	SI (Solo lectura)
Clientes http:	5	5

AUTÓMATA S7-1500

El PLC S7-1500 es el autómata más reciente creado por Siemens. Salió al mercado en el año 2013. Se pretende que sea el sustituto de los actuales S7-300/400 a medio plazo.

El PLC S7-1500 rompe con todo lo anterior, empezando por su aspecto exterior. Dispone de una pequeña pantalla desde la que se puede obtener información importante sin necesidad de utilizar ningún software. En la Figura 5 se muestra el PLC 1516-3 PN/DP. Otra de sus mejoras es todo lo referente a las redes de comunicación, tan relevante en la actualidad. Su formato de programación también cuenta con mejoras evidentes tendentes a la simplificación y la estandarización.

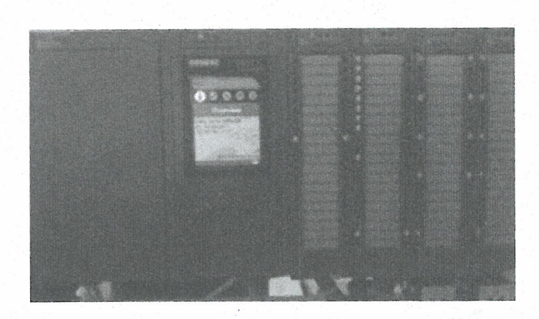


Figura 5

Un aspecto importante es que el nuevo PLC puede ejecutar las instrucciones de sus antecesores (S7-300/400). Esto hace que el paso de un PLC a otro no intimide a los actuales programadores de S7-300/400, pero tiene muchas nuevas instrucciones.

Algunas de las características de estos PLC son:

- ✓ Mayor rendimiento del sistema.
- ✓ Funcionalidad Motion Control integrada.
- ✓ PROFINET IO IRT (tiempo real isocrónico).
- ✓ Pantalla integrada para el manejo y diagnóstico a pie de máquina.
- ✓ Innovaciones de lenguaje STEP 7 manteniendo las funciones probadas.

La principal característica externa del PLC S7-1500 es su pantalla. En ella se ofrece información de la configuración del autómata y también se puede hacer un diagnóstico sin necesidad de ningún *software*. El perfil sobre el que se instala el PLC y los diferentes módulos también son diferentes a los de la familia S7-300. Este perfil permite la conexión de hasta 32 módulos. Los diferentes módulos y el PLC van unidos por la parte trasera con un bus en forma de U, de forma similar al que hay en las series S7-300.

El autómata más reciente (noviembre 2016) que ha sacado SIEMENS es el S7-1500T, un PLC preparado para los nuevos tiempos de la cuarta revolución industrial. En la industria estas innovaciones se deben plasmar en la denominada «Industria 4.0».

En la Figura 6 se ha esquematizado la configuración básica de un autómata S7-1500.

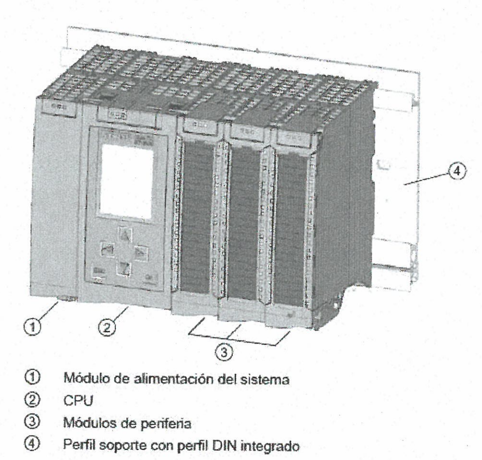


Figura 6

En la siguiente tabla se detallan algunas de las características de los autómatas S7-1500.

CPU	1512C-1 PN	1516-3 PN/DP	1516F-3 PN/DP
Memoria de trabajo			
Integrada (para programa)	250 kbyte	1 Mbyte	1,5 Mbyte
Integrada (para datos)	1 Mbyte	5 Mbyte	5 Mbyte
Display			
Diagonal de la pantalla	3,45 cm	6,1 cm	6,1 cm
Interfaces			
Nº interfaces PROFINET	1	2	2
Nº Interfaces PROFIBUS		1	1
Contaje y medida			
Funciones de contaje	SI	SI	SI
High Speed Counter Integrados	6	-	-
Tiempo e ejecución en la CPU			
Para operaciones en bit	48 ns	10 ns	10 ns
Funciones de seguridad	NO	NO	SI

En la Figura 7 se puede observar cómo se coloca la U sobre la CPU y esta sobre el perfil. Primero se coloca el perfil en la parte trasera de la CPU (1), después se debe enganchar la CPU sobre el perfil soporte (2) y finalmente se atornilla en la parte inferior (3).

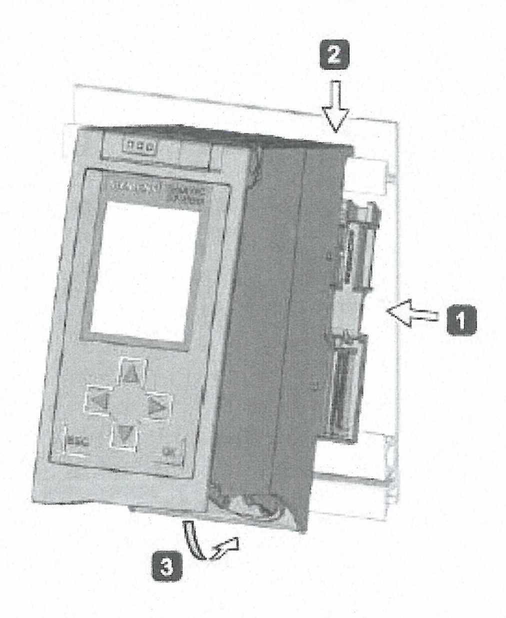


Figura 7

En la Figura 8 puede verse el *display* del PLC S7-1500. En él se obtiene una valiosa información, simplemente navegando por sus menús. En la parte superior se puede observar el estado del PLC y debajo los diferentes submenús. También muestra la denominación de la CPU y su referencia. Además de dar información relevante, permite modificar valores desde el *display*, tales como la dirección y la máscara de subred.

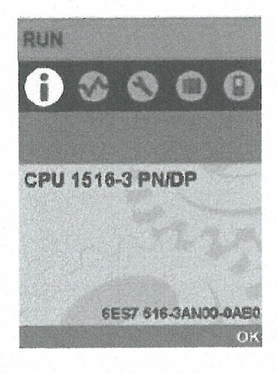


Figura 8

En la siguiente tabla se recogen los diferentes submenús:

Comandos de menú principales	Significado	Significado
	Sinopsis	El menú **Vista general** contiene datos sobre las propiedades de la CPU.

Comandos de menú principales	Significado	Significado
	Diagnóstico	El menú **Diagnóstico** contiene datos sobre los avisos de diagnóstico, descripciones de diagnóstico e indicadores de alarma. Asimismo, informa sobre las propiedades de red de cada interfaz de la CPU.
	Configuración	El menú **Configuración** permite asignar las direcciones IP de la CPU, ajustar la fecha, hora, zona horaria, el estado operativo (RUN/STOP) y los niveles de protección, ejecutar el borrado total de la CPU, restablecer la configuración de fábrica del equipo y consultar el estado de las actualizaciones de firmware.
	Módulos	El menú **Módulos** contiene datos acerca de los módulos utilizados en su configuración. Los módulos pueden estar agregados como módulos centralizados o descentralizados. Los módulos descentralizados están conectados a la CPU vía PROFINET o PROFIBUS. Aquí se ofrece la posibilidad de ajustar las direcciones IP para un CP.
	Display	En el menú **Pantalla** se realizan los ajustes de la pantalla, p. ej., el idioma, el brillo y el modo de ahorro de energía (en el modo de ahorro de energía se oscurece la pantalla; en el modo de reserva, se apaga la pantalla).

4. CONFIGURACIÓN DEL AUTÓMATA EN STEP7 Y TIA PORTAL (V13)

Como se ha mencionado anteriormente, tanto los PLC de la serie S7-300 como su entorno de programación STEP 7 V5.5 o anteriores son elementos que en un futuro próximo desaparecerán. Actualmente (noviembre 2016) siguen en vigor y no están descatalogados por SIEMENS. Hay muchos dispositivos *hardware*, tal como PLC, unidades de E/S y otros, que ya no se pueden utilizar en la nueva plataforma de TIA PORTAL. Esto ocurre con todos los dispositivos anteriores a 2007. Si se desea hacer alguna modificación en estos elementos, solo hay dos opciones: o se cambian por otro nuevo o se realizan las modificaciones con STEP 7 V5.5 o anteriores. Por tal motivo, aquí se tratan las dos plataformas, STEP/V5.5 o anteriores y TIA PORTAL.

La programación es común para los dos tipos de PLC, al menos la que se va a tratar en este libro.

En esta primera parte se va a configurar el PLC en el *software* Step 7 V5.5. También se utilizará el simulador que servirá para realizar los programas del libro. Todas las acciones de configuración serán las mismas que si se utilizara el autómata real.

CONFIGURACIÓN CON STEP 7 V5.5

Con este entorno de programación solo se pueden utilizar PLC S7-300. Para los S7-1500 es necesario utilizar la plataforma TIA PORTAL.

Step 7 arranca accionando el icono Administrador SIMATIC. Este entorno tiene varias ventanas donde configurar y programar. La primera que aparece nada más iniciar el programa se llama Simatic Manager (Administrador Simatic). Si se desea arrancar y activar el simulador, debe pulsarse el icono en la parte superior de Administrador Simatic. Ahora no se hará nada con el simulador, solo arrancarlo para poder configurar correctamente el PLC.

Antes de empezar a programar, es necesario que el formato de nuestro sistema, CPU, fuente de alimentación y módulos de entrada/salida estén identificados en nuestro Step 7. Una vez configurado el sistema en el administrador, se cargará a la CPU (al simulador). En este administrador también se utilizará el editor del programa para, posteriormente, transmitirlo a la CPU y ejecutarlo o, en su defecto, simular. Para realizar todo esto, debe indicarse al administrador de qué elementos disponemos.

El orden que se debe seguir es el siguiente:

1. Abrir el **Administrador Simatic**.

   Esta es la pantalla que aparece al abrirlo:

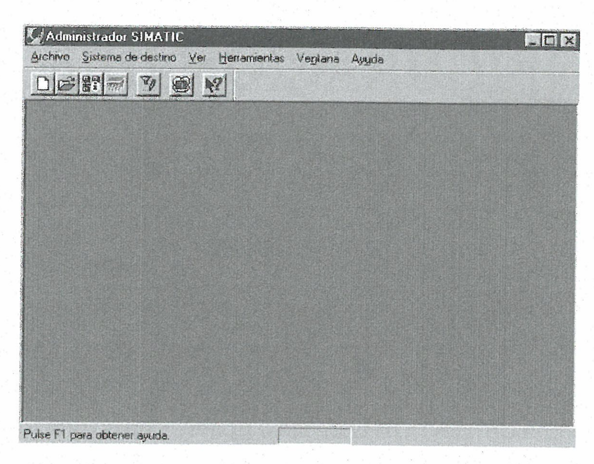


Figura 9

2. Pulsando en el icono **Nuevo** (🗋), iniciamos nuevo proyecto, le asignamos nombre y aceptamos:

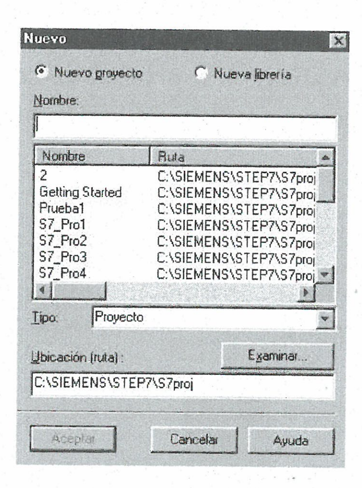


Figura 10

3. Tocando sobre la carpeta del proyecto y pulsando el botón derecho del ratón se seleccionan del menú desplegable las siguientes opciones (Figura 11):

Insertar nuevo objeto /Equipo SIMATIC 300

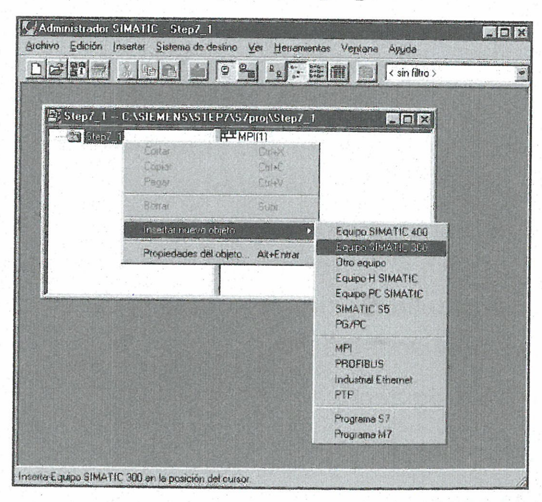


Figura 11

4. Con el ratón (botón izquierdo) pulsando dos veces sobre **Hardware.**

   También se puede acceder al *hardware* pulsando con el botón derecho del ratón en **SIMATIC 300(1)** y seleccionando del menú desplegable la opción **Abrir objeto.**

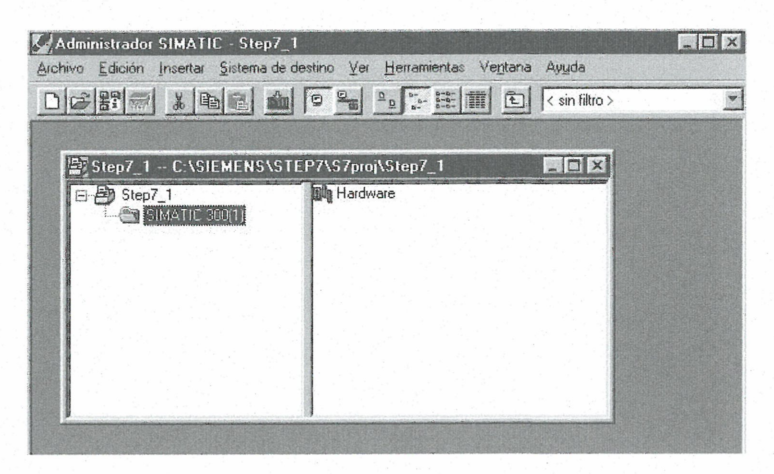


Figura 12

5. En cualquiera de los dos casos aparece la siguiente ventana:

   Esta es la ventana de configuración del *hardware* (**HW Config**). Aquí es donde se colocan todos los elementos de los que dispone el PLC, Figura 13.

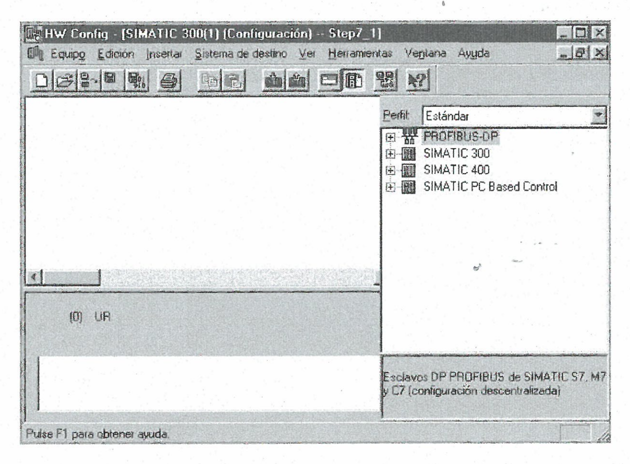


Figura 13

   A la derecha se sitúa el catálogo que se puede hacer mostrar u ocultar con el icono **Catálogo** . Si alguno de los componentes no aparece en el catálogo, habrá que actualizarlo, por ejemplo, desde la página web de Siemens. En un tema posterior se explicará cómo hacerlo.

6. Ahora se deben ir colocando cada uno de los elementos de nuestro sistema según su disposición real sobre el bastidor.

Lo primero será colocar el perfil (bastidor) sobre el que se situarán los componentes. A continuación se sigue con la fuente de alimentación (PS), CPU y tarjetas de entrada/salida (SM) tanto digitales como analógicas.

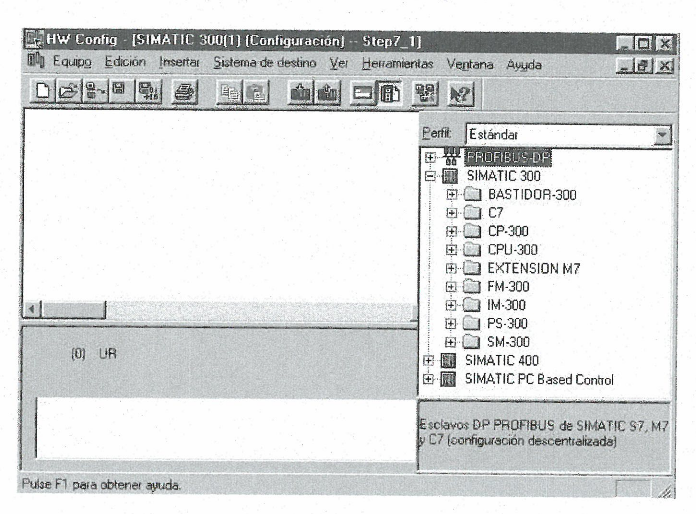


Figura 14

Para colocar el bastidor se debe pulsar dos veces sobre **Perfil soporte**. Una vez seleccionado el bastidor, aparecerá la ventana que se muestra en la Figura 15.

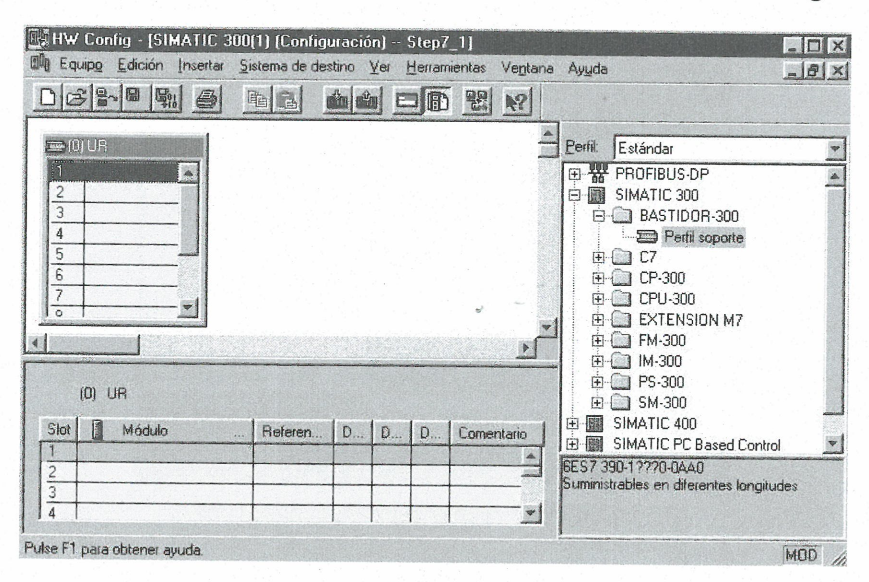


Figura 15

Para la fuente de alimentación, se selecciona la que corresponda a las características de la que dispongamos. Se busca en la carpeta del catálogo denominada **PS** pulsando dos veces con el ratón (botón izquierdo). La fuente escogida se colocará automáticamente sobre la posición n.º 1 del bastidor. En el caso de trabajar con el simulador, se puede colocar cualquiera de ellas.

La Figura 16 indica cómo queda la ventana una vez seleccionada la fuente de alimentación.

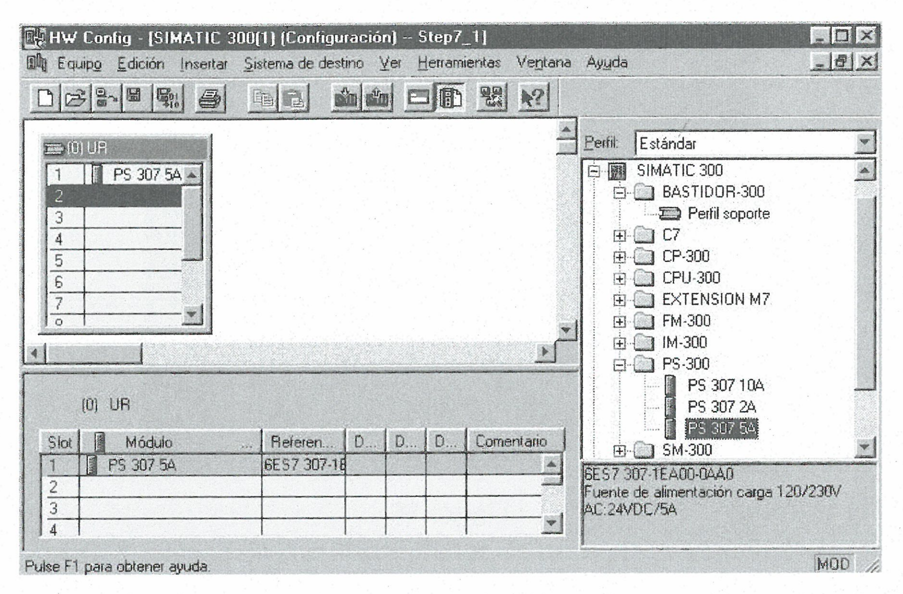


Figura 16

Seleccionar la **CPU** según la **referencia** del autómata. Al pulsar sobre ella se situará en la posición 2. Para el simulador, seleccionar una CPU 315 2 DP, como aparece en la Figura 17.

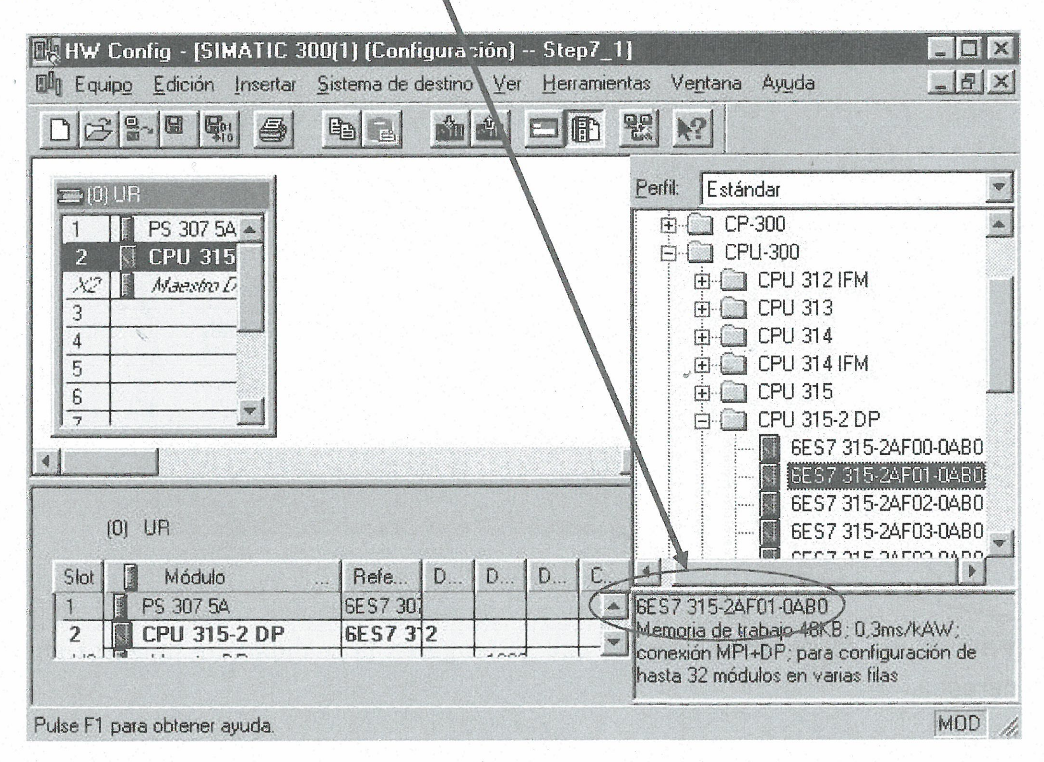


Figura 17

Si se selecciona una CPU que disponga de salida para conexión a red Profibus o Profinet (p. ej., 315-2 DP), aparecerán mensajes referentes a dichas redes. Si no se dispone de esas redes, o no se desea configurar la red en ese momento, se aceptarán todos los mensajes que vayan apareciendo sin modificar nada de su contenido.

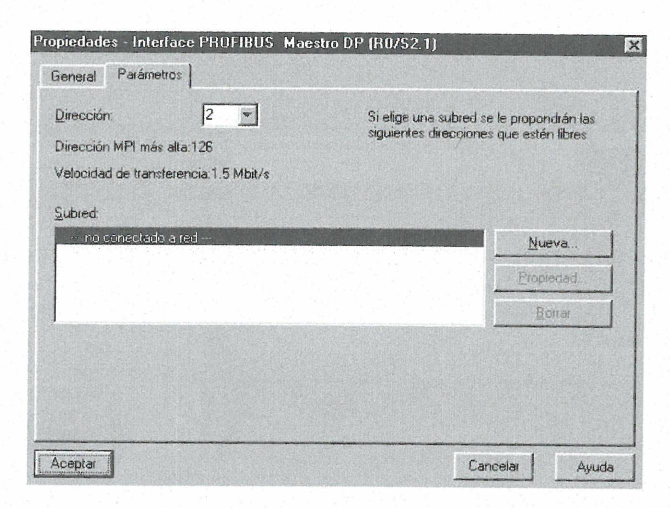


Figura 18

La tercera posición del bastidor se deja libre si no se utiliza ningún módulo de ampliación de bastidor (**IM**), como es en este caso.

A partir del *slot* 4 se seleccionan los diferentes módulos de E/S (**SM**) de los que se disponga, tanto digitales como analógicos.

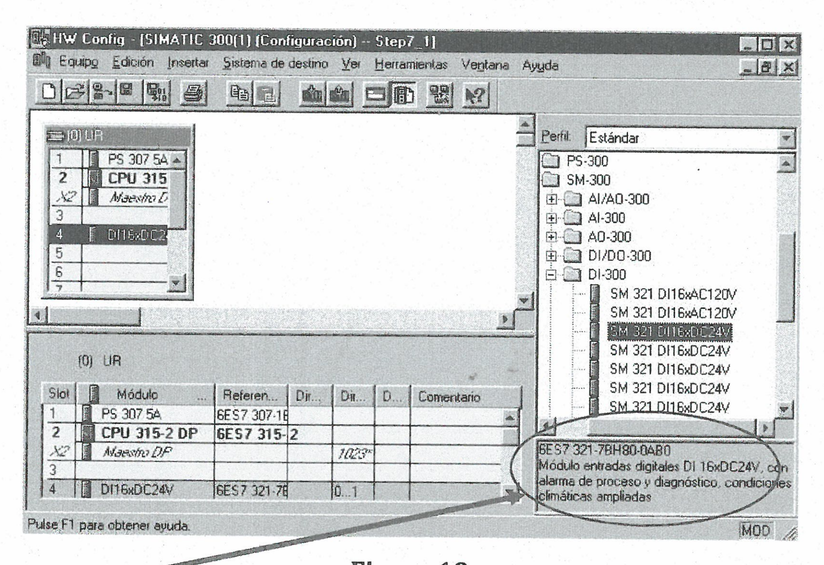


Figura 19

La **referencia** debe ser la misma que la que se tenga en la realidad. Se muestra en la parte inferior derecha.

Si se va a utilizar el simulador, se puede poner cualquier referencia. En ese caso se aconsejan las siguientes referencias:

DI/DO: Entradas salidas digitales 6ES7 323-1BL00-0AA0

AI/AO: Entradas salidas analógicas............ 6ES7 334-0CE01-0AA0

Con esto concluye la configuración. Seguidamente se sale de la configuración del *hardware* y se graba cuando pregunte si no se ha hecho previamente.

Nota: Todo este proceso de situar los componentes del sistema en el bastidor se puede realizar arrastrando el módulo con el ratón desde el catálogo a la ranura correspondiente del bastidor, en vez de usar la doble pulsación.

Una vez finalizada la configuración, se debe cargar en la CPU. Para ello, se debe activar el icono de carga. Esto se puede hacer desde la ventana de configuración del *hardware* o desde el administrador situándose en la carpeta SIMATIC 300(1). Si se utiliza el simulador, lo cargará sobre él, pero debe estar abierto previamente porque de lo contrario dará error de comunicación al no disponer de ningún autómata real.

Si hubiera algún problema, se puede ir a **Ajustar Interface,** que se encuentra en el menú **Herramientas del Administrador Simatic**. Aquí se selecciona **PC Adapter (MPI)** o, mejor, **PC Adapter (Auto).** Si se trabaja con el simulador, se debe elegir **PLC SIMUL**, el tipo Profibus, por ejemplo. Esta es la interfaz que conecta el ordenador con el autómata para poder enviar la configuración y los programas, y que se muestra en las Figuras 20 y 21.

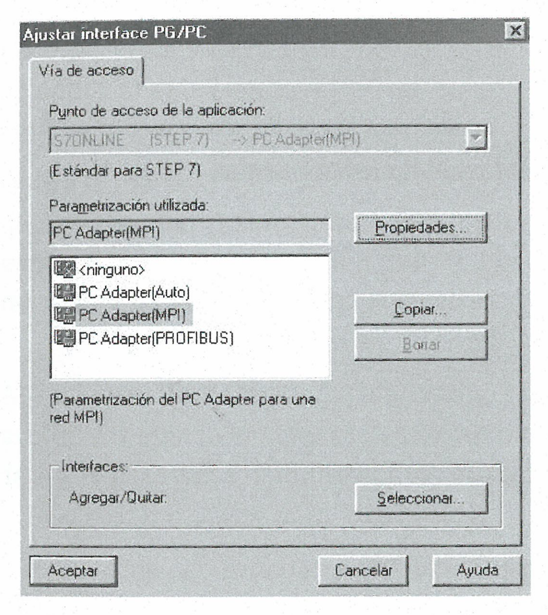


Figura 20

Figura 21

Con estas operaciones ya se tiene el autómata preparado para poder programarlo.

CONFIGURACIÓN CON TIA PORTAL V13

En el año 2009, Siemens revolucionó el mundo industrial de los procesos automáticos sacando al mercado su plataforma software TIA PORTAL. Con este entorno solo se necesita una herramienta de programación, todo se hace desde el mismo software. Por ejemplo, no es necesario un entorno para STEP 7 y otro para el WINCC por separado, para programar un PLC y una pantalla gráfica. Ahora todo se integra en la misma plataforma. Eso sí, si se desea trabajar con pantallas gráficas será necesario disponer del software WinCC, y su correspondiente licencia, pero se integrará en TIA PORTAL. Al instalarlo se instala por defecto Step7, herramienta para programar.

La actual versión de TIA PORTAL es la V14. Esta nueva versión (octubre 2016) integra novedades con respecto a la comunicación OPC y la nube. En este libro se utiliza la versión 13.

Se va configurar un autómata S7-300 y posteriormente un PLC 1500, siempre con TIA PORTAL. También se puede utilizar para los autómatas de la serie S7-1200.

Configuración con TIA PORTAL de un PLC S7-300

Cuando se abre TIA PORTAL, por defecto aparece la **ventana del portal**. Desde ahí se debe iniciar el proyecto y configurar el PLC que se va a utilizar. En la Figura 22 se puede ver esa primera ventana.

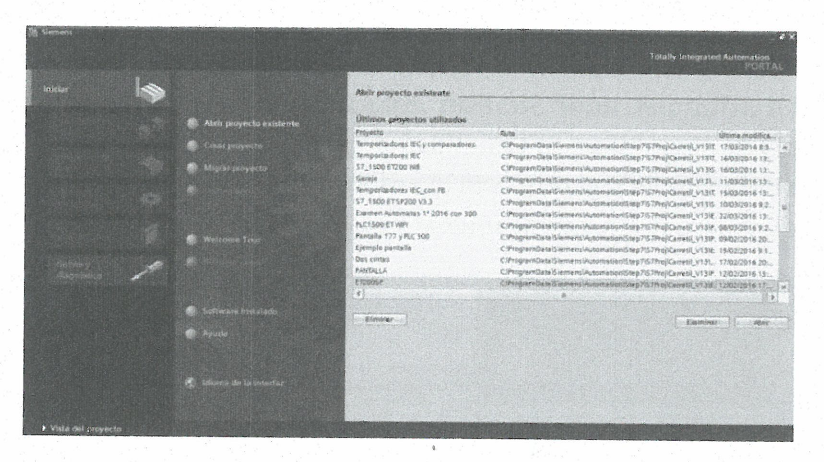


Figura 22

Para empezar, hay varias opciones: abrir un proyecto ya existente, crear uno nuevo o migrar un proyecto existente en alguna de las plataformas antiguas de STEP 7 a la nueva de TIA PORTAL.

Desde esta ventana también se pueden buscar las actualizaciones que pudiera haber del TIA PORTAL y otros programas instalados. Esto es muy importante para estar al día dentro de una determinada versión. Se accede desde **Software instalado**, donde también se puede consultar el *software* que ya se tiene cargado.

Se crea un nuevo proyecto indicando su nombre y accionando **Crear**. Aparece la Figura 23:

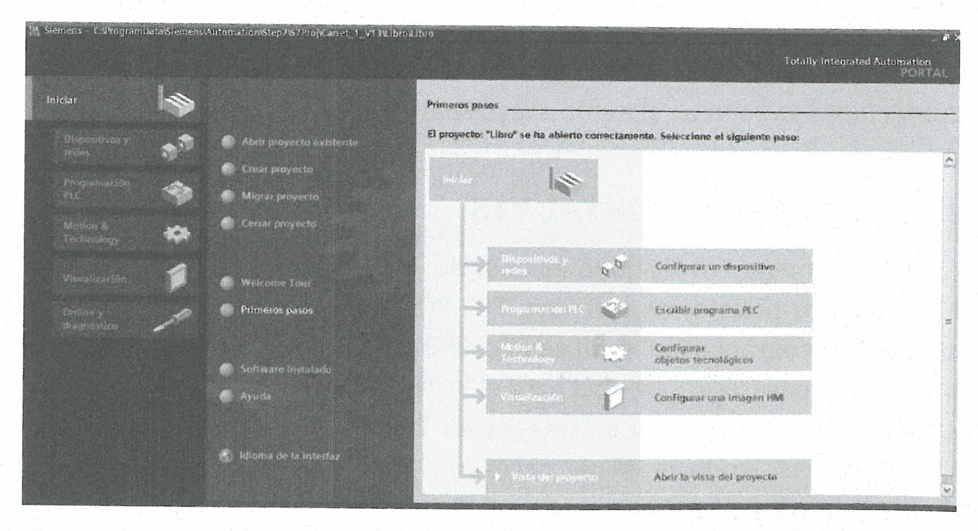


Figura 23

Lo siguiente que se deberá hacer es **Configurar un dispositivo** y **Agregar un dispositivo**, tal como se indica en las Figuras 24 y 25.

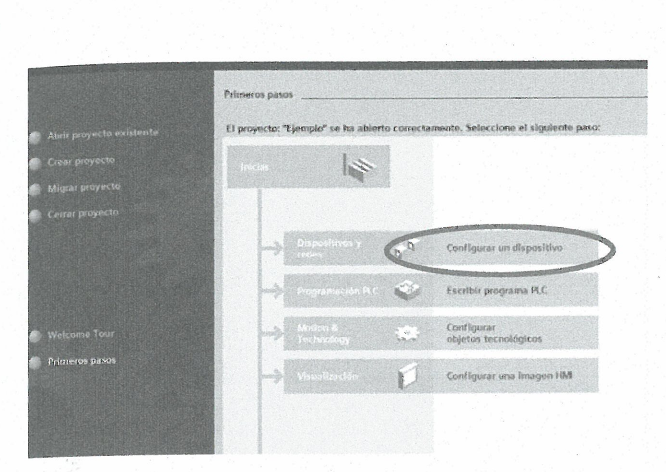


Figura 24

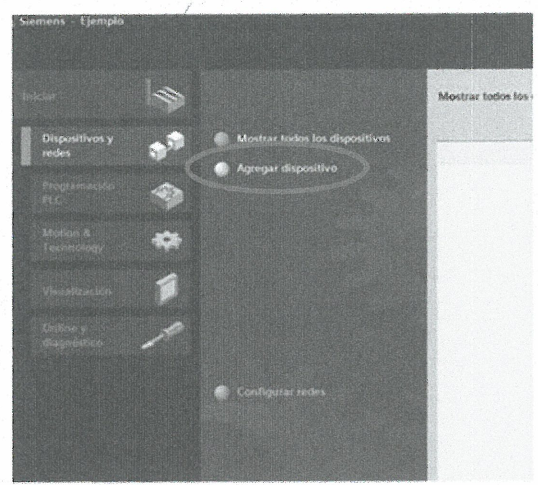


Figura 25

Es importante saber que no todos los PLC se van a poder utilizar con TIA PORTAL, aunque se actualice el catálogo. Eso se debe a problemas de compatibilidad y a la política de SIEMENS (solo dispositivos que Siemens no haya descatalogado hasta el 1 de octubre de 2007).

Desde la pantalla que aparece al agregar un dispositivo, se debe seleccionar el dispositivo del que se dispone o seleccionar **CPU 300 sin especificar**. Se ha de seleccionar **CPU 300 sin especificar** y, a continuación, pulsar **Agregar**. Este procedimiento no es válido si se utiliza el simulador. Para el simulador se empleará otro método que se explica más adelante.

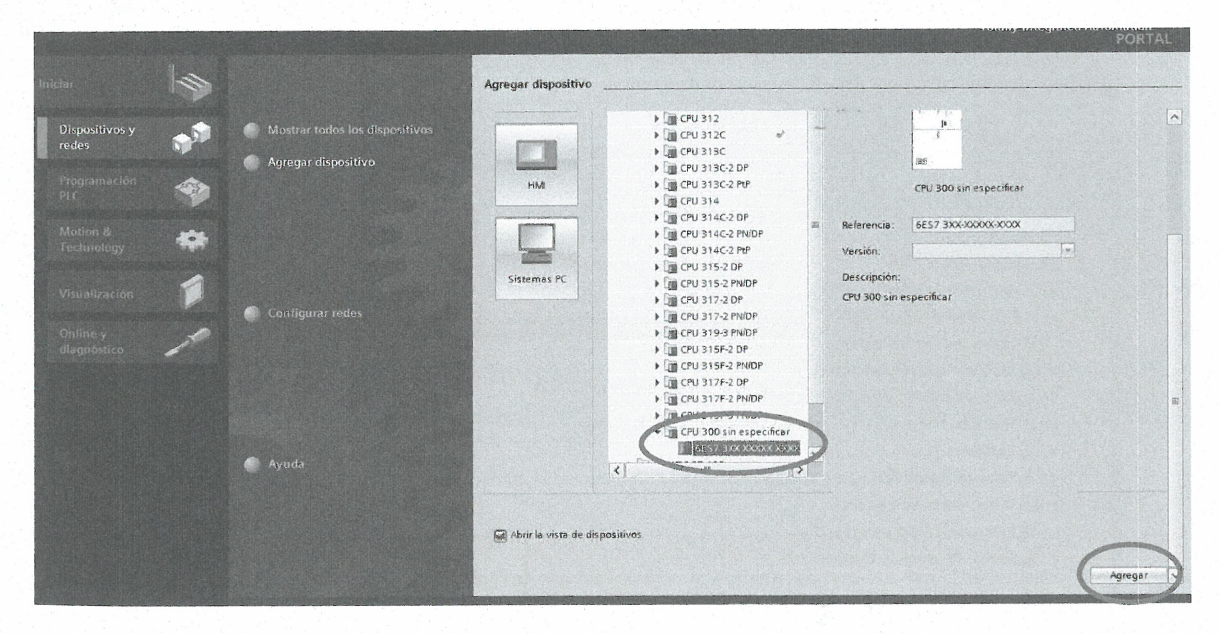


Figura 26

Aparece la ventana que se representa en la Figura 27.

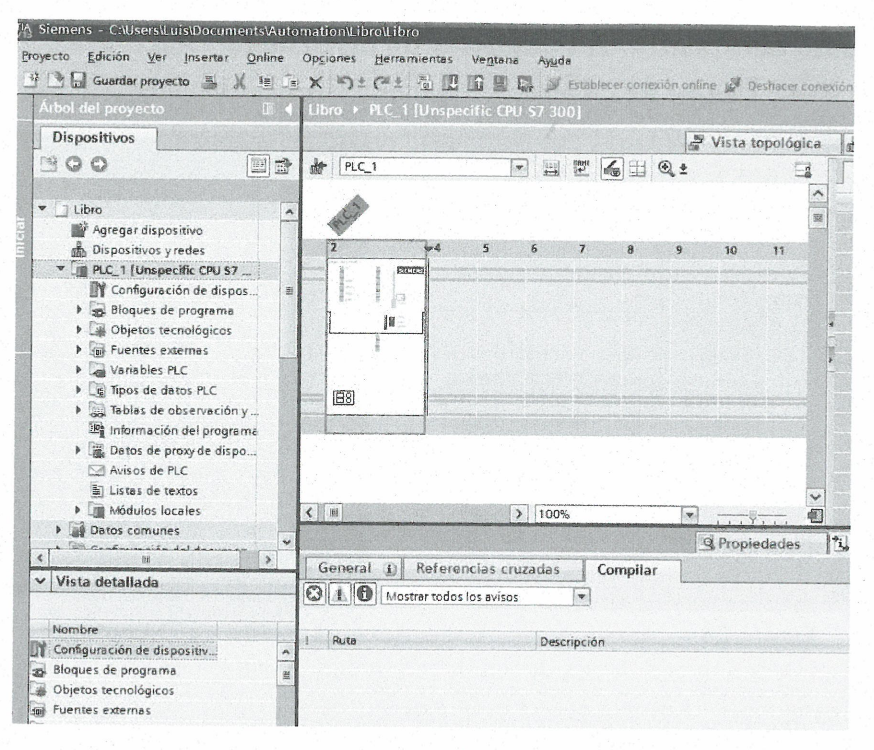


Figura 27

Para las CPU 300, desde esa ventana se accede al menú **Online** y allí se selecciona **Carga del dispositivo como estación nueva (hardware y software)**. Previamente se debe hacer clic sobre la CPU para que nos aparezca esta opción activada, tal como se aprecia en la Figura 28.

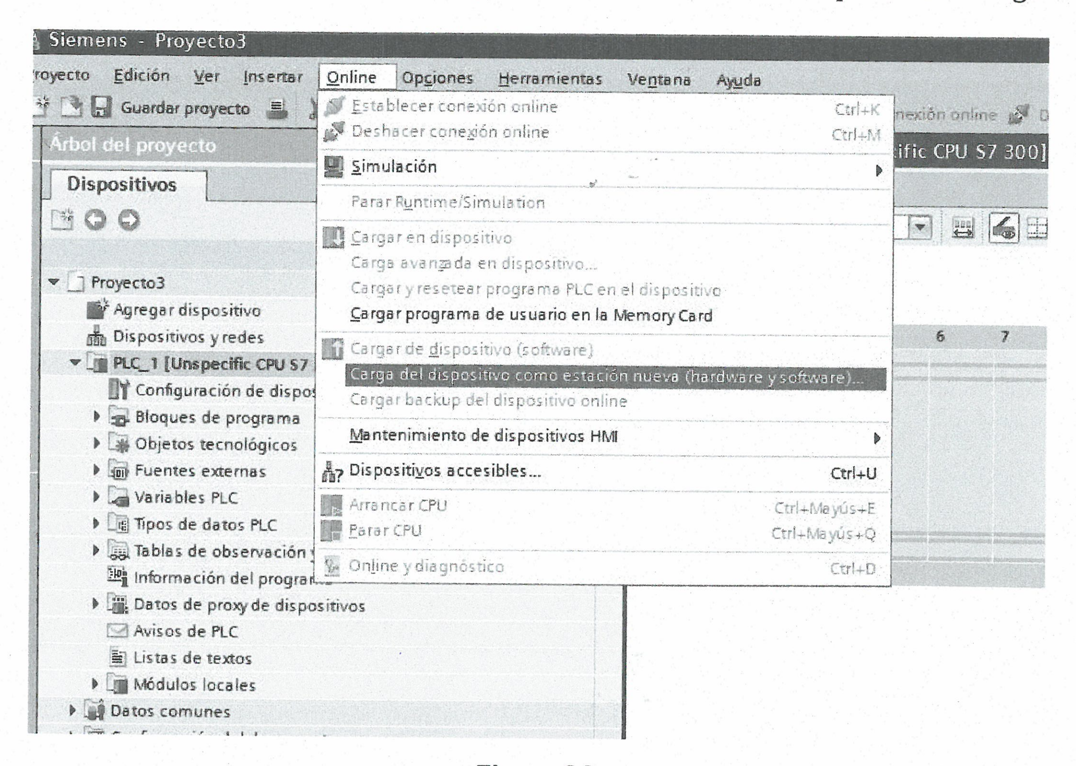


Figura 28

Ahora, y en la V13 de TIA PORTAL, comienza la búsqueda de la CPU conectada. Cuando la detecta, nos lo indica, y pulsamos **Cargar**, tal como indica la Figura 29.

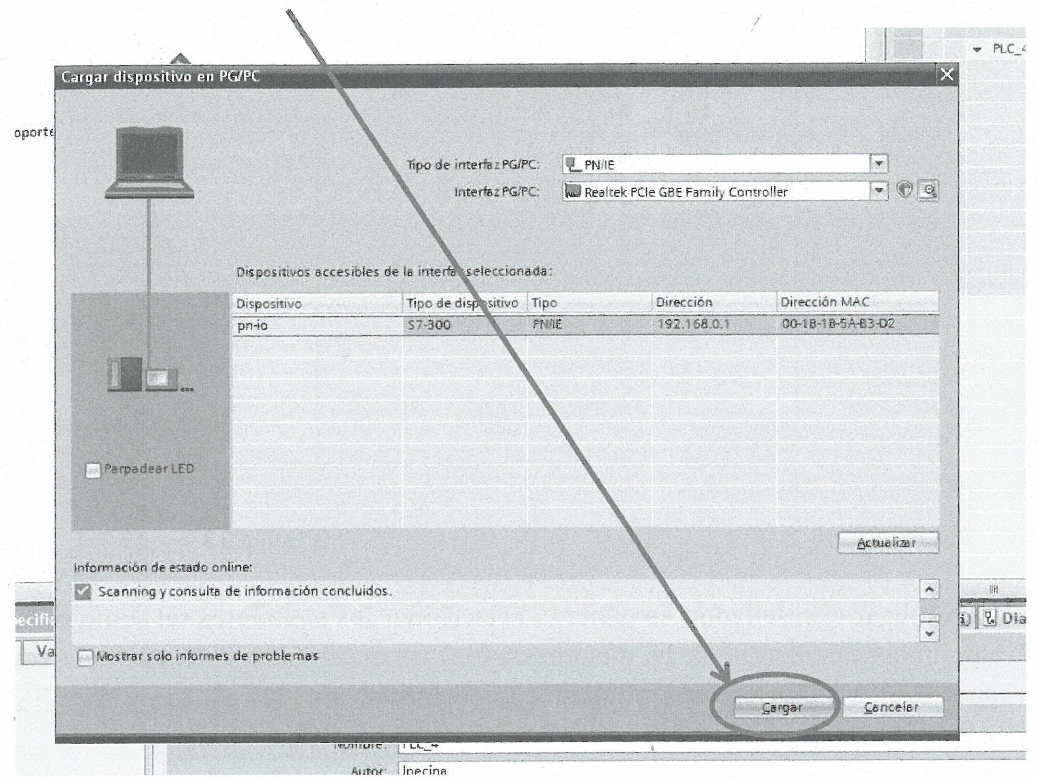
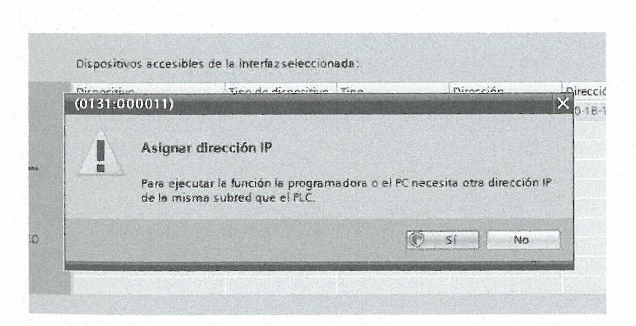


Figura 29

Si aparecen alguna de las dos ventanas siguientes, debemos seleccionar **Sí** y **Aceptar** (Figuras 30 y 31).



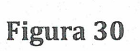


Figura 30

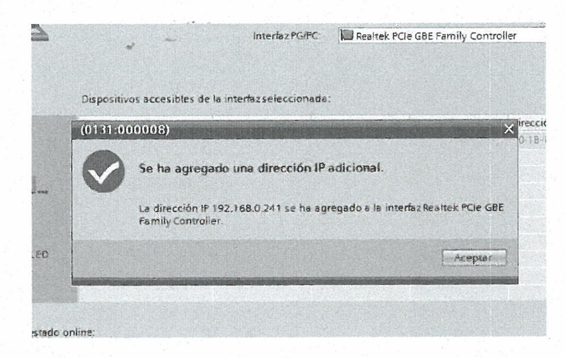


Figura 31

En la parte del árbol del proyecto aparecerá el PLC cargado, según se muestra en la Figura 32. Ahora hemos de pulsar sobre **Configuración de dispositivos** y aparecerá el PLC y su *hardware* completo, como se ve en la Figura 33.

De esta forma se ha completado la configuración del sistema PLC con el que se desea trabajar.

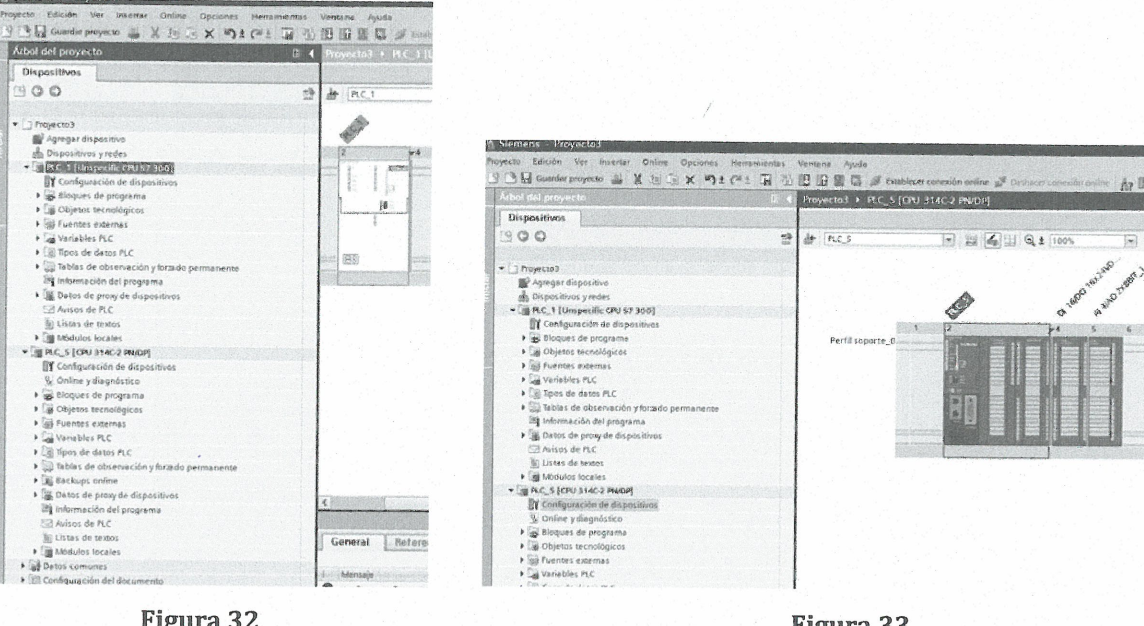


Figura 32 Figura 33

En el caso de utilizar el simulador, se deberán seleccionar los elementos tal como se ha hecho en STEP 5.5. En la pantalla que se ha elegido un PLC sin especificar, ahora se debe concretar uno, da igual cuál, ya que se va a simular (por ejemplo, un PLC S7-135-2PN/DP). Ver la Figura 34.

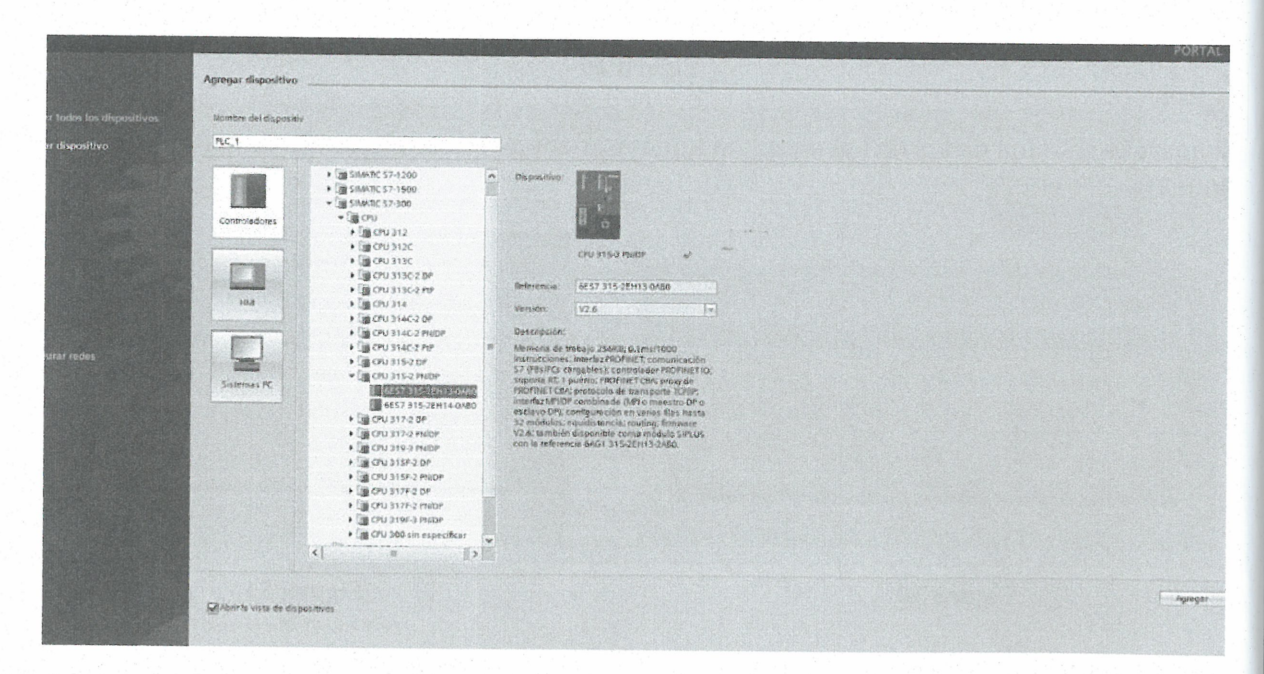


Figura 34

Una vez que aparece la vista de proyecto con el PLC elegido, faltan por completar los distintos módulos que debe llevar, como entradas y salidas. Estos se eligen del **Catálogo** que se encuentra a la derecha, tal como se hacía en STEP7 5.5. En la Figura 35 se puede ver cómo queda una vez completado. Ahora se debe enviar esta configuración al PLC real (o simulado) y para ello se activa el icono **Cargar en dispositivo.** Antes hay que abrir el **Simulador.**

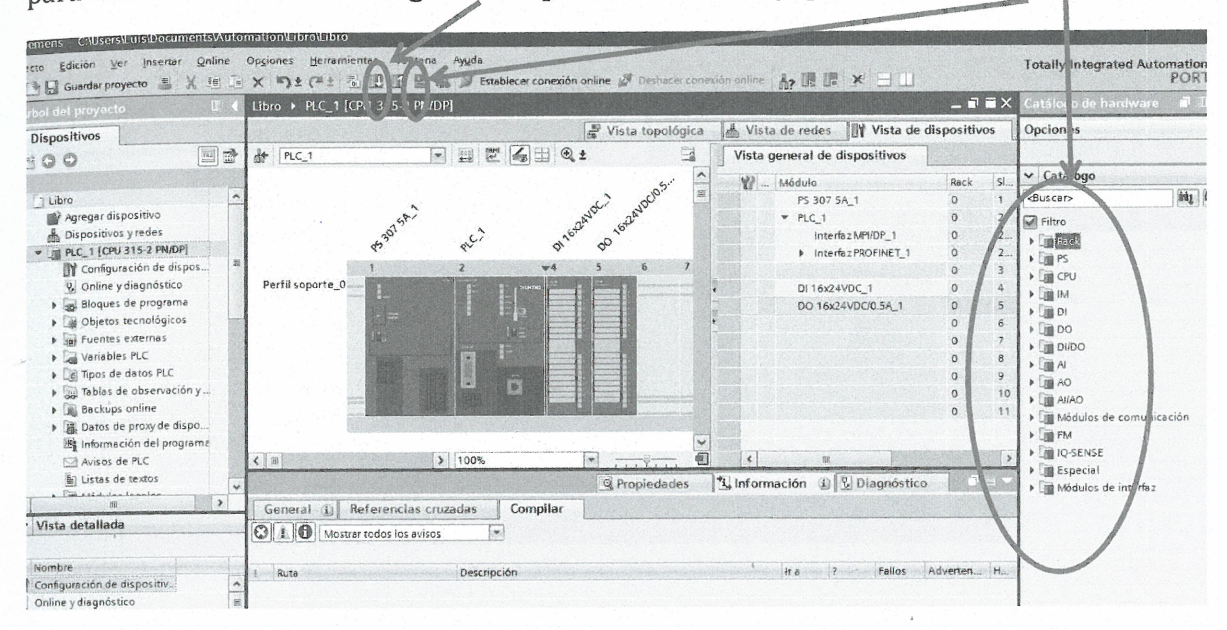


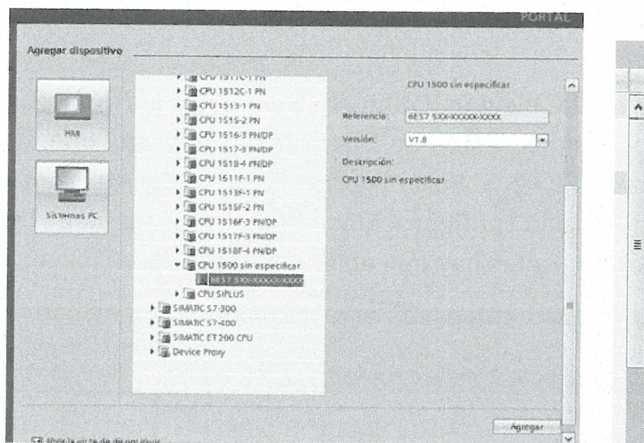
Figura 35

Al cargar sobre el PLC, saldrán las ventanas que ya habían aparecido anteriormente. Son las Figuras 29, 30 y 31. Además, en el siguiente punto se incide sobre el envío y carga en el PLC.

Configuración con TIA PORTAL de un PLC S7-1500

Para configurar un PLC S7-1500 se empieza de igual forma que con el S7-300. Una vez que se ha llegado a la opción de *"Configurar un dispositivo"* y *"Agregar un dispositivo"*, debe seleccionarse el PLC deseado o indicar *"CPU 1500 sin especificar"*. Se va a ver la opción de *"CPU 1500 sin especificar"*. A continuación pulse *"Agregar"*. La figura A representa esta situación.

En la siguiente ventana (figura B) accione *"Determinar la configuración del dispositivo conectado"*. Aparecerá una pantalla donde se debe indicar el tipo de *interface* que se está utilizando para comunicar con el PLC. En la figura C se puede apreciar lo que se pide para poder conectar con el PLC y absorber de él la configuración que tiene. Este método simplifica la configuración del sistema, ya que en tres pasos se carga la estación.



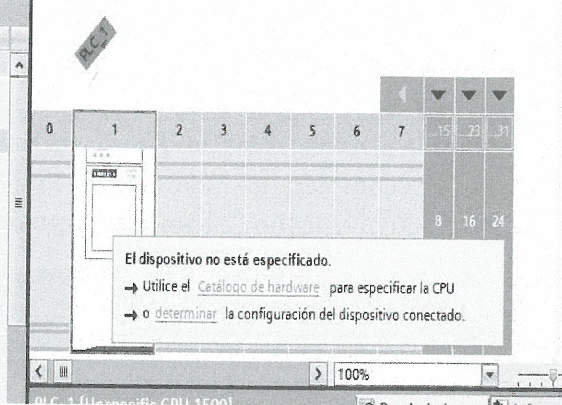


Figura A **Figura B**

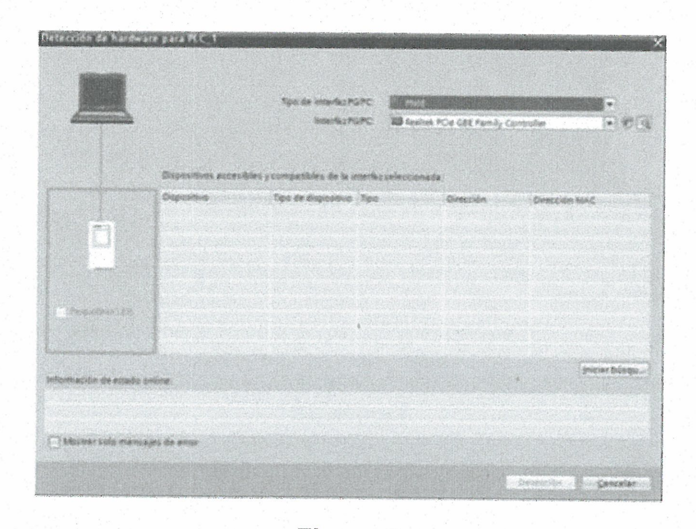


Figura C

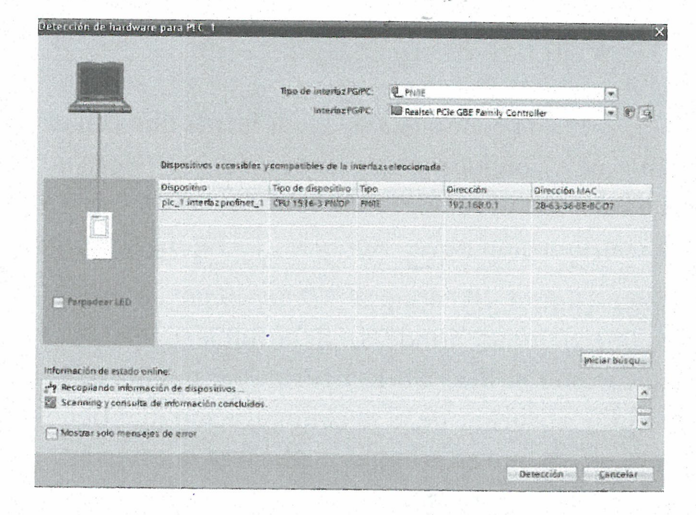


Figura D

Accionar *"Iniciar búsqueda"* y al cabo de poco tiempo aparecerá la figura D; después se podrá activar *"Detección"* (esto puede tardar un poco más en aparecer activo). Este procedimiento solo es válido con un PLC de la serie S7-1500. Se debe tener presente varias cosas. En la figura

D, si se activa la pestaña de *"Parpadear Led"*, el autómata parpadeará alguno de sus ledes (situación muy útil cuando se dispone de más de un PLC en red).

Otra cuestión a tener muy en cuenta es qué sucede si no carga el PLC o se deja algo a cargar. Esto ocurre cuando el PLC o alguna de las E/S que dispone no se encuentran en el catálogo del TIA PORTAL. Para solucionar este problema será necesario actualizar el TIA PORTAL. Si no se ha hecho, se deberá actualizar el catálogo a la última versión de los ficheros de actualización (HSP). En el capítulo de actualización de catálogo se trata este importante asunto.

Si todo ha ido bien, aparecerá algo similar a la figura E aunque dependerá del tipo de PLC y módulos de E/S de que se disponga.

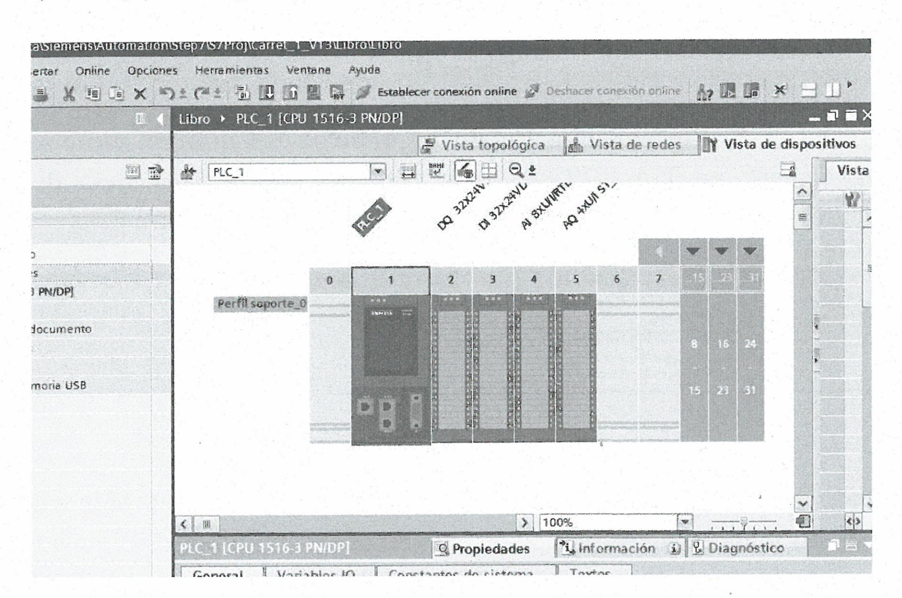


Figura E

A la hora de transmitir pueden salir las ventanas de las figuras 30 y 31, vistas anteriormente.

Con esto ya se está en disposición de poder programar el PLC S7-1500. Naturalmente, también puede configurarse este autómata con el método clásico de buscar los dispositivos desde el catálogo.

ACTUALIZACIÓN DEL CATÁLOGO

Una de las primeras cosas que se debe saber es que no todos los dispositivos (antiguos) van a poder ser utilizados en TIA PORTAL, aunque actualice el catálogo, debido a problemas de compatibilidad. Solo dispositivos que Siemens no haya descatalogado hasta el mes de octubre de 2007 serán compatibles. Si no son compatibles se deberá utilizar STEP 7 versión 5.5 o inferiores.

Para actualizar el catálogo hay tres caminos. Uno es comprobar que TIA PORTAL esté actualizado a la última versión. Si no es así, se debe actualizar. Para ello, se puede ir a la aplicación *"Automation Software Updater"*. También se puede acceder desde TIA PORTAL, en la vista del portal, como se aprecia en la figura F.

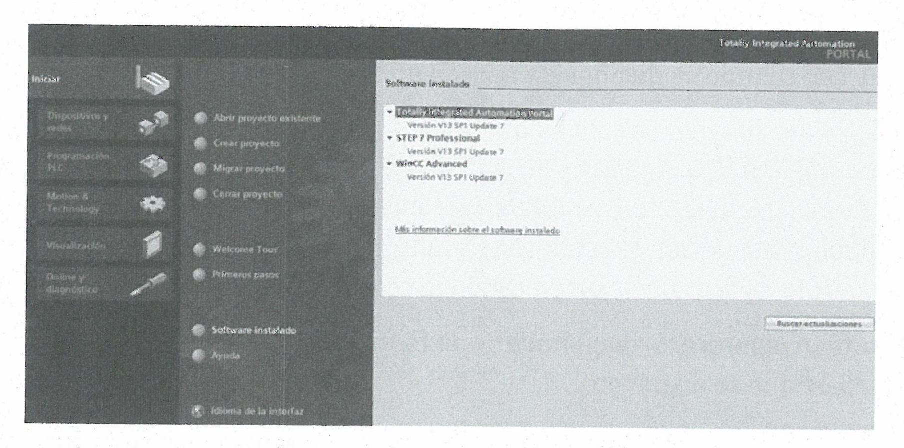


Figura F

La segunda opción es mediante la instalación de los archivos *hsp,* que son como se llaman a los ficheros de actualización en TIA PORTAL. Para ello, hay que ir a la vista del proyecto y seleccionar Opciones/ Support Packages. La ruta se puede ver en la figura G.

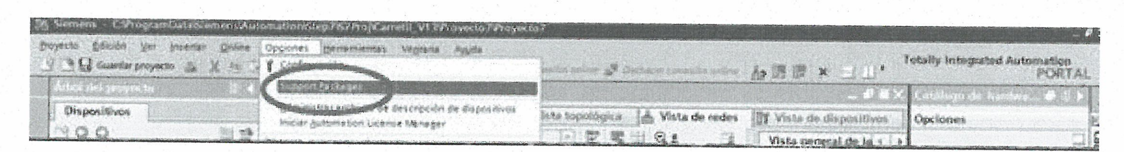


Figura G

Aparece la ventana reflejada en la figura H (puede que las diferentes filas salgan vacías o distintas a la imagen) y allí se activa la opción de descargar de Internet. Previamente hay que crearse una cuenta en la web de Siemens. Se guarda el fichero en el PC y se debe descomprimir en el lugar donde lo haya dejado (descargas).

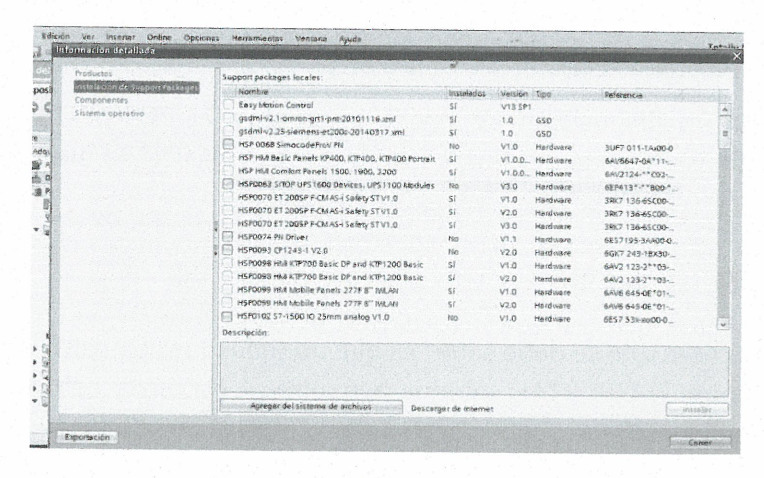


Figura H

A continuación, se selecciona el archivo guardado desde *"Agregar del sistema de archivos".* Se seleccionan los dispositivos que se quieren instalar.

La tercera y última opción es instalar los ficheros *gsd* como se hacía en versiones anteriores de STEP 7. Se debe buscar el fichero *gsd* previamente y después, desde *"Opciones/Administrar archivos de descripción de dispositivos"*, buscar el fichero e instalar.

ENTORNO DE LA PLATAFORMA TIA PORTAL

A continuación se va a hacer una descripción general de la plataforma TIA PORTAL, además de lo ya visto hasta ahora. Una vez que se está en la vista de proyecto, se puede observar que aparece mucha información. Toda la necesaria se encuentra en esta pantalla. El problema es que son demasiadas ventanas y se tendrá que navegar por todas. Es muy importante disponer de un monitor de tamaño considerable para trabajar con comodidad, de lo contrario se hace un poco pesado cambiar entre las distintas ventanas. Hay que tener en cuenta que muchas de estas ventanas tienen información que no se ve en principio si no nos desplazamos por ellas. De lo contrario, podemos perder datos importantes. Se debe utilizar el desplazamiento (*scroll*) de las ventanas. En la Figura 36 se incluye el conjunto de las diferentes ventanas.

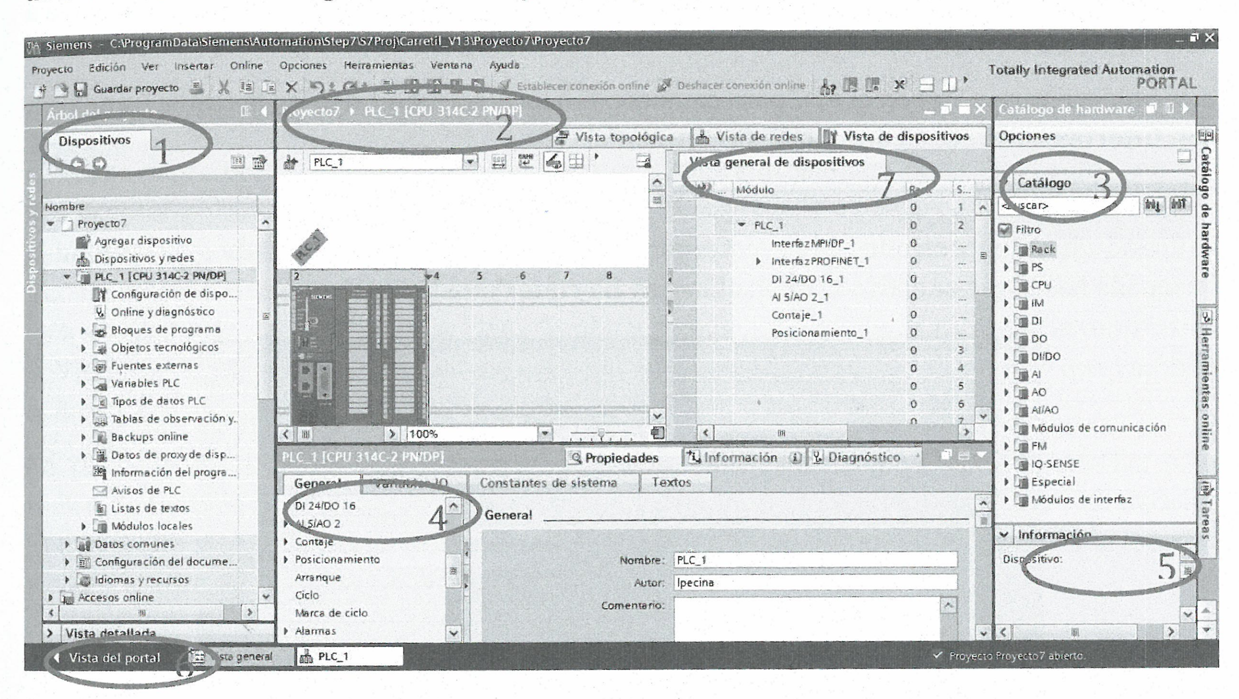


Figura 36

Por una parte, a la izquierda aparece el árbol del proyecto (1) con todos los dispositivos que lo forman. En la vista detallada (6) se indican las diferentes opciones de lo que se seleccione en la parte superior. Aquí se podrán agregar más dispositivos.

En la zona de *hardware* del proyecto (2) se puede comprobar todo el *hardware* del proyecto. Existen tres opciones: vista topológica, vista de redes y vista de dispositivos. En la vista de dispositivos se debe terminar de configurar la parte de *hardware* que nos falta, mientras que la vista de redes se emplea para configurar las redes de comunicación industrial. En la vista topológica se aprecian todos los componentes *hardware* del proyecto. En esta ventana también consta la información sobre los módulos dispuestos en el proyecto desde la vista general de dispositivos (7). En la parte inferior aparecen las propiedades (4) de los dispositivos que se seleccionen en la ventana superior de *hardware*. Aquí se deben indicar las direcciones de red o también consultar las direcciones que tienen cada tarjeta de E/S, y otras características. A la derecha se encuentra el catálogo de *hardware* (3) y abajo (5) la

información sobre el dispositivo seleccionado en el catálogo. Aquí es muy importante poder ver la versión del dispositivo y utilizar para ello el desplazamiento de la ventana *(scroll).*

Cada vez que se desee cargar en el PLC la configuración *hardware* o *software*, se debe pulsar en el icono **Cargar en dispositivo** .

En la Figura 37 se muestra la ventana que aparece cuando se va a transmitir. Los pasos a seguir son:

1. Seleccionar el tipo de interfaz PG/PC.

2. Indicar la interfaz utilizada. Si está abierto el simulador, se debe utilizar la interfaz del simulador como indica la pantalla. Si hay un PLC real, se selecciona en la interfaz que se vaya a utilizar, MPI, Profibus o Profinet.

3. Si hubiera varios Slot en el PLC para comunicar, habría que indicarle, en la casilla **Conexión con Interfaz/subred,** en qué Slot está conectado.

4. Iniciar búsqueda.

5. Cuando haya encontrado el PLC ....Cargar. Puede tardar un instante desde que termina la búsqueda hasta que se activa la casilla **Cargar**. Hay que darle tiempo.

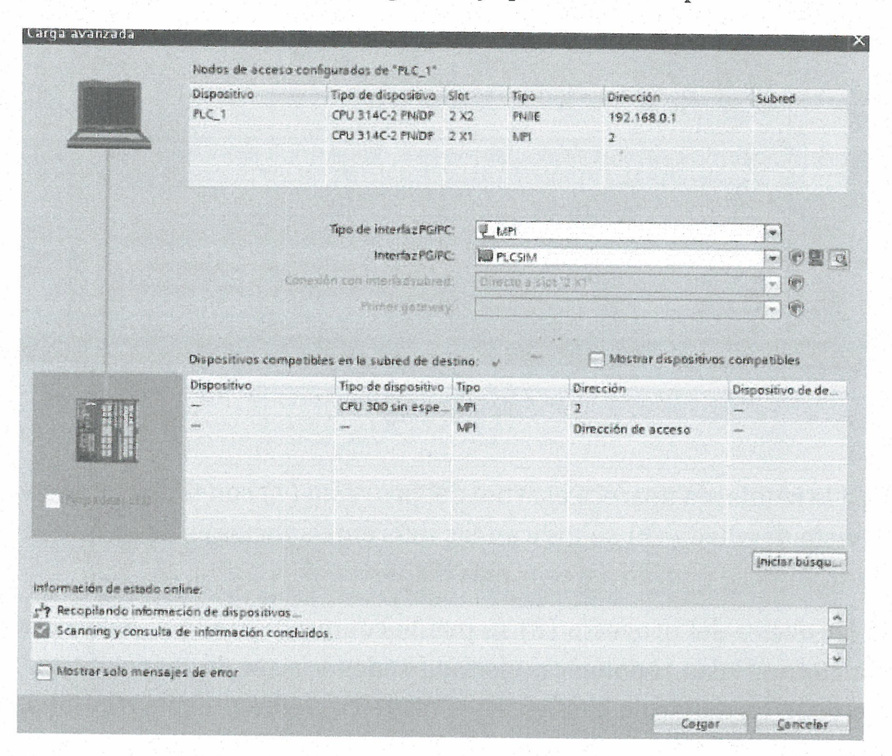


Figura 37

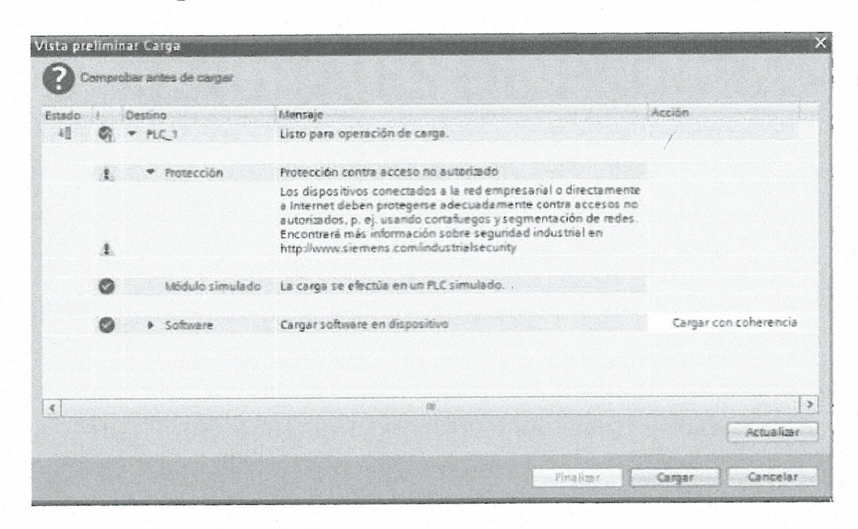


Figura 38

La carga sobre el PLC comenzará de inmediato y saldrá la pantalla de la Figura 38, que hay que aceptar pulsando **Cargar**. Si es necesario (en el caso de hacer redes de comunicación), se deberán dar direcciones a los diferentes dispositivos del proyecto. Para ello, se debe tocar el PLC o dispositivo que se desee direccionar y abajo en las propiedades buscar la interfaz que se va a utilizar para cambiar su dirección. También se puede tocar con el ratón en la interfaz de la **Vista de dispositivos** y abajo ya saldrán las propiedades de la interfaz, en las que cambiar la dirección. Esto se aprecia en la Figura 39.

En dicha figura se puede observar que el PLC seleccionado dispone de una interfaz para realizar dos tipos de comunicación: MPI o Profibus, en el caso de un PLC S7 300/400. Para los PLC S7 1500 no hay opción MPI. Desde aquí se pueden cambiar las propiedades de la interfaz y elegir el **tipo de comunicación.**

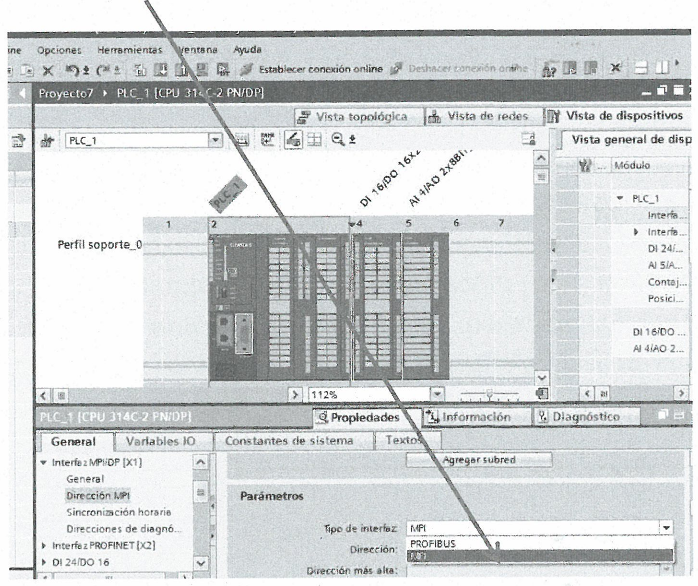


Figura 39

Ahora ya se está en disposición de poder trabajar con el PLC parar realizar su programación.

5. LENGUAJES DE PROGRAMACIÓN INSTRUCCIONES DE BIT

INTRODUCCIÓN

En este capítulo y en los siguientes se va a estudiar el modo de programar el autómata. Los autómatas de las series 300 y 400 son compatibles en cuanto a *software* con los de la serie 1500. Son tres tipos de lenguajes los que existen para programarlos: KOP, AWL y FUP. Vamos a utilizar mayoritariamente el AWL. Este lenguaje tiene más posibilidades y es más cómodo de utilizar. Además de estos tres lenguajes hay otro, que se denomina SCL, y que es un lenguaje de «alto nivel» de tipo estructurado, basado en Pascal.

Los autómatas de la serie S7 1200 no se pueden programar en AWL, pero sí en el resto de lenguajes. También tienen compatibilidad de *software* con los S7 1500.

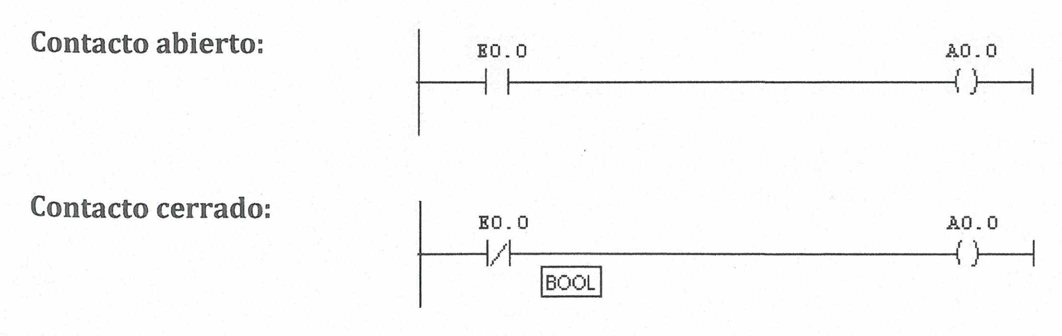
En este tema también se aprenderá a utilizar el editor de programas y el simulador. El simulador es el mismo para STEP 7 que para TIA PORTAL V13, para los autómatas de la serie S7 300.

TIPOS DE LENGUAJES DE PROGRAMACIÓN

Como ya se ha destacado, son tres los tipos de lenguajes que se utilizan para programar los autómatas de las series 300/400 y 1500. Inicialmente se verán los aspectos de formato diferenciadores.

KOP

Es un lenguaje más intuitivo y el favorito de aquellos que están más acostumbrados al diseño clásico de automatismos cableados. Es un lenguaje de contactos. Un uno lógico será un contacto abierto y un contacto cerrado será un cero lógico. Lo dicho puede parecer contradictorio pero, si pensamos en la acción sobre los contactos, vemos que al activar/accionar un contacto abierto, se permite el paso de corriente por él. Y, por el contrario, accionar un contacto cerrado lo abre y no permite el paso de corriente.



Un ejemplo de un programa en KOP de tres contactos en serie sería este:

E0.0 E0.1 E0.2 A0.0

AWL

Este lenguaje es de tipo texto y tiene más posibilidades. Parece más complicado para los que no están acostumbrados a programar.

Los contactos se representan mediante la operación lógica con la que se conectan al circuito. Si el contacto es abierto, se coloca solo la orden de la operación lógica y, si es cerrado, se añade a dicha operación lógica una N (de negación).

Un contacto abierto............... U E0.0

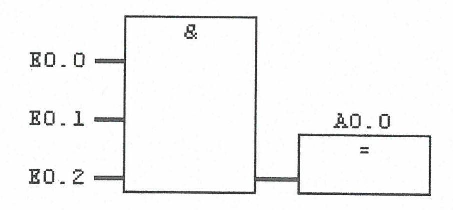
Un contacto cerrado.............. UN E0.0

El ejemplo de tres contactos en serie, en AWL sería de este modo:

```
U    E    0.0
U    E    0.1
U    E    0.2
=    A    0.0
       BOOL
```

FUP

Este lenguaje es de tipo gráfico, dibujo de módulos. No es muy utilizado. El ejemplo sería en FUP:



En este libro se va a utilizar el lenguaje AWL. Los primeros programas se harán utilizando KOP para, posteriormente, pasarlos a AWL.

INSTRUCCIONES DE BIT

Las instrucciones de bit son aquellas que manejan exclusivamente operandos a nivel de bit. Las operaciones que se pueden realizar con bits son solo lógicas. Las analógicas trabajan a nivel de palabras.

RLO (Resultado lógico del programa)

Como se sabe, las instrucciones en un programa se ejecutan una detrás de otra de forma secuencial. Existe un bit en el registro de estado que nos indica el valor lógico del programa después de cada operación lógica al ejecutar órdenes que modifican el valor de este bit. Al iniciar un programa, el RLO tendrá el valor de la primera orden que lo modifique. A esta acción se le denomina *primera consulta*.

Instrucción de asignación

La primera orden que se va a estudiar es la *ASIGNACIÓN.* Esta orden pasa el valor del RLO al operando indicado en la orden.

La instrucción es =

Por ejemplo .... = **A0.0** Esta orden pondrá la salida (bit) A0.0 a uno cuando el RLO sea uno, y a cero cuando sea cero.

Es muy importante destacar que **NUNCA** se pueden **repetir dos asignaciones iguales**, es decir, que si ya existe una asignación a la salida A0.0, no deberá parecer otra a la misma salida. En este caso se pondrían en paralelo, como veremos más adelante. Este un uno de los principios que no deben olvidarse.

Instrucción Y (and)

La instrucción AND realiza la operación lógica AND entre el RLO y el operando que aparece en la instrucción. La instrucción es **U.**

Ejemplo:

```
U E0.0
U E0.1
= A0.0
```

En este caso la entrada E0.0 está en serie con la entrada E0.1. Dicho de otro modo, la E0.0 hace la operación lógica AND con el valor del RLO en ese instante. Como es la primera orden, el RLO toma el valor de la entrada E0.0 (primera consulta). La entrada E0.1 hace la operación AND con el valor del RLO, que ha sido modificado por la entrada E0.0. El valor final del RLO se asigna a la salida A0.0. Resumiendo, el valor de las dos entradas en serie se asigna a la salida.

En KOP el programa anterior sería de esta forma:

E0.0 E0.1 A0.0

Instrucción O (or)

La instrucción OR realiza la operación lógica OR entre el RLO y el operando que aparece en la instrucción. La instrucción es **O.**

Ejemplo:

```
U E0.0
O E0.1
= A0.0
```

El RLO toma el valor de la entrada E0.0 por ser primera consulta. Es por eso que, en esa primera orden es lo mismo poner U que O, aunque suele ser normal poner U. La entrada E0.1 hace la operación OR con el valor del RLO, que ha tomado el valor de la entrada E0.0. El valor final del RLO se asigna a la salida A0.0. Resumiendo, el valor de las dos entradas en paralelo se asigna a la salida.

En KOP el programa anterior sería de esta forma:

E0.0 A0.0
E0.1

Tanto en U como en O, si el contacto fuera cerrado, la orden en AWL sería **UN** y **ON** respectivamente, y en KOP sería un contacto cerrado E0.0 puesto en serie o en paralelo.

Otras instrucciones con el RLO

Además de las instrucciones vistas anteriormente, donde se modificaba el valor del bit RLO en función de los resultados de las operaciones lógicas ejecutadas, existen otras órdenes que actúan sobre el RLO.

Estas son:

- NOT: invierte el valor del RLO.
- SET: pone a uno el estado del RLO.

- CLR: pone a cero el RLO.
- SAVE: almacena el RLO en el registro de estado. Puede volver a leerse con la instrucción BR.

SIMULADOR

A partir de este momento ya se pueden empezar a realizar sencillos ejercicios en KOP y en AWL utilizando estas tres instrucciones aprendidas. Para ello, se debe utilizar el simulador y por ese motivo comentaremos brevemente cómo utilizarlo. El simulador para los autómatas S7 300 es el mismo tanto en TIA PORTAL como en STEP 7. El simulador para el PLC 1500 es diferente y se explicará al final de este libro.

Lo primero que hay que hacer es abrirlo. En STEP 7 hay que activarlo mediante el icono que se encuentra en la parte superior derecha del administrador Simatic: . En TIA PORTAL es el icono . Una vez abierto, encontraremos lo siguiente (Figura 40).

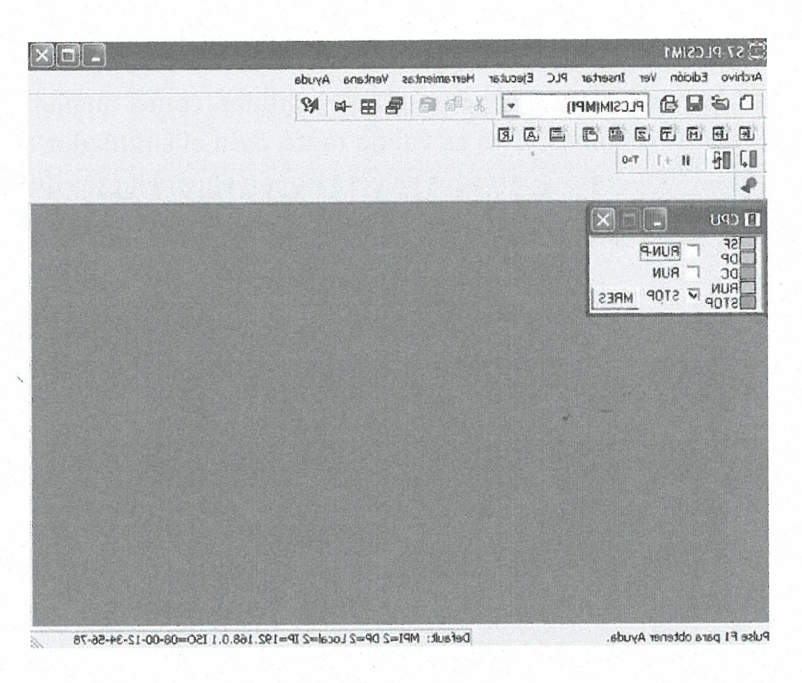


Figura 40

Se observa que aparecen los ledes y el conmutador del PLC. Los ledes indican:

SF: error fatal / DP: conexión profibus, si la hay / DC: alimentación del PLC / RUN: cuando el PLC está ejecutando el programa / STOP: cuando el PLC está parado.

El conmutador indica:

STOP: para parar el PLC / RUN: el programa se ejecuta y no podrá cambiarse / RUN-P: el programa se ejecuta y puede cambiarse mientras se ejecuta / MRES: reset y borrado total. Cuando se activa este pulsador, hay que volver a cargar el *hardware*.

Botones para visualizar Entradas, Salidas, Marcas, Temporizadores, Contadores...... Pulsando esos botones se van colocando las entradas, salidas, temporizadores......... Se arranca el simulador (RUN-P) y se van activando las entradas según convenga para ver el funcionamiento del programa. En las salidas se puede ver cuál es su comportamiento y comprobar si es el correcto según el enunciado.

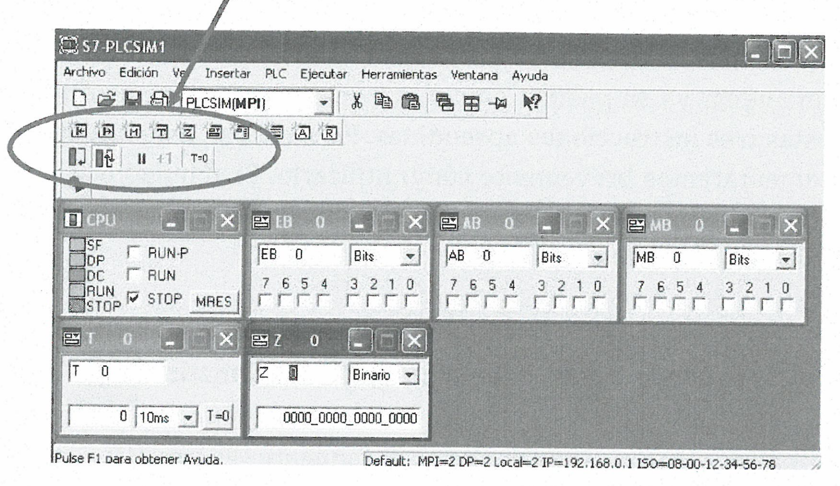


Figura 41

Para poder utilizar las entradas y salidas, tanto digitales como analógicas, es necesario conocer cuáles son sus direcciones. Esto es válido tanto para el simulador como para cuando tengamos el PLC. En los siguientes apartados se verá dónde encontrar las direcciones.

STEP7 5.5

Cuando se ha configurado el PLC, el STEP 7 ha dado unas direcciones a cada E/S. Por ello deberemos ir a **Hardware** para conocer sus direcciones. En la Figura 42 se indican esas direcciones:

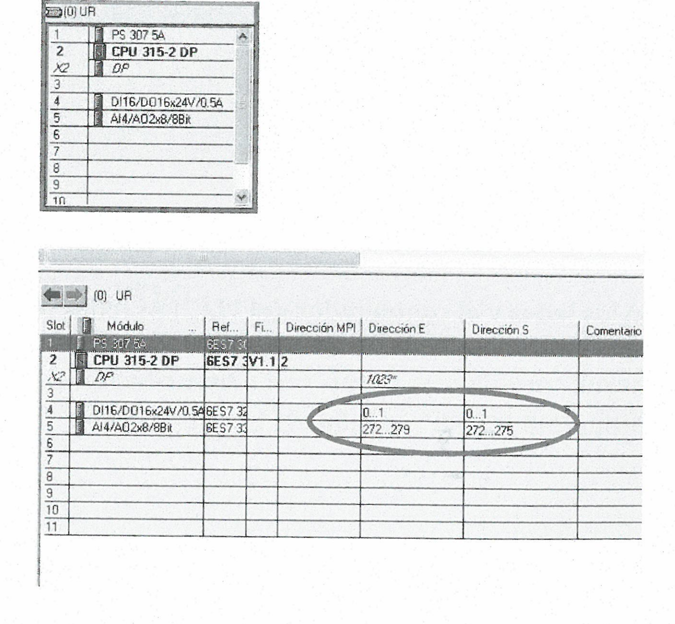


Figura 42

Según se ve en la Figura 42 las direcciones para las entradas digitales son el byte 0 y el byte 1, lo mismo que para las salidas digitales. Es decir, que las direcciones, a nivel de bit, serán:

Entradas:
E0.0, E0.1........................ E0.7
E1.0, E1.1 .... E1.7

Salidas:
A0.0, A0.1 .... A0.7
A1.0, A1.1 .... A1.7

Para seleccionar diferentes direcciones de entradas o salidas en el simulador, se debe cambiar la dirección sobre la entrada o salida. Por defecto, cada vez que se coloca una E/S, la dirección es la cero. Para cambiarla simplemente se borra el 0 y se pone la dirección que se desee, tal como indica la Figura 43.

Aquí se cambia la dirección.

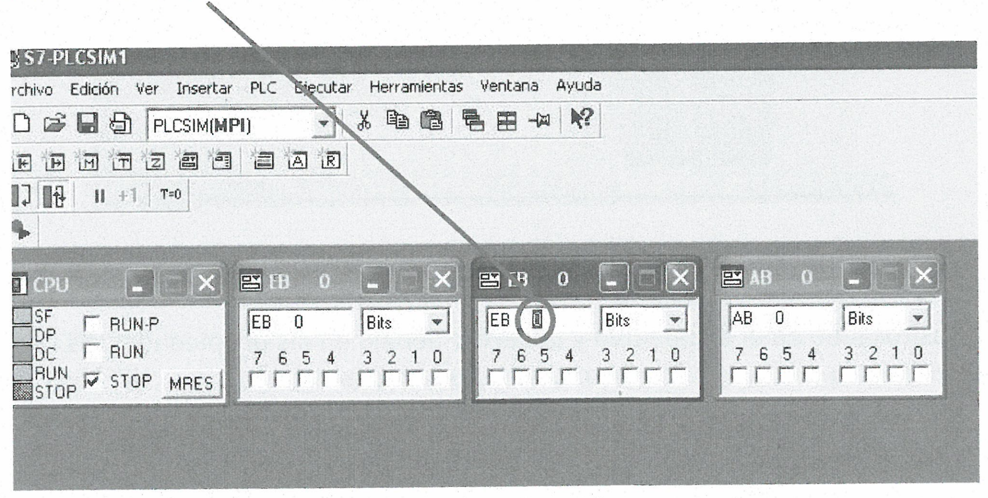


Figura 43

Para sacar los acumuladores, se debe pulsar el icono A, tal como se indica en la Figura 44.

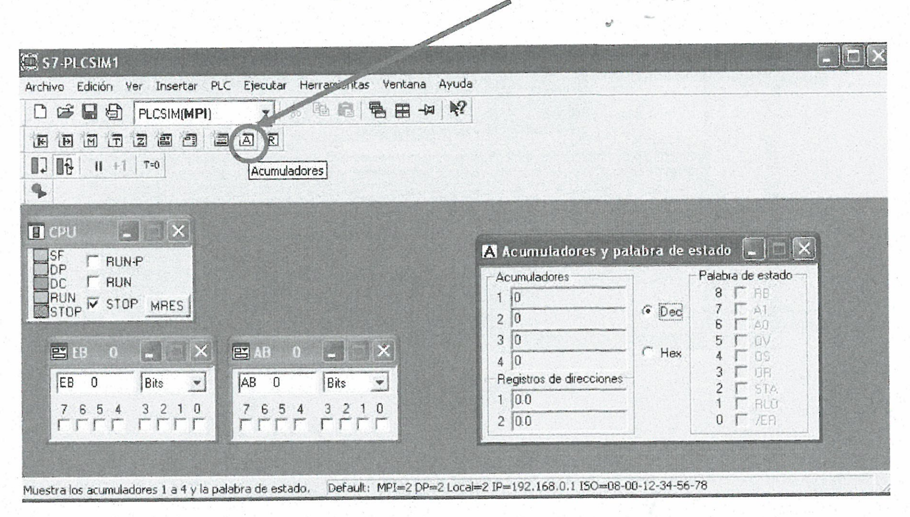


Figura 44

Para conocer las direcciones de las tarjetas de E/S en TIA PORTAL hay varios procedimientos. Uno es consultar en la vista general de dispositivos (Figura 45).

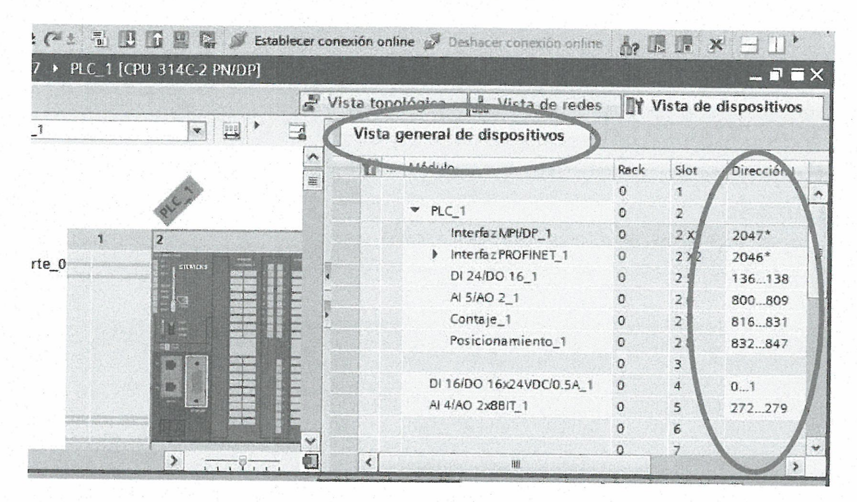


Figura 45

Otra forma es tocando en el dispositivo y observar debajo en las propiedades. Es lo que indica la Figura 46.

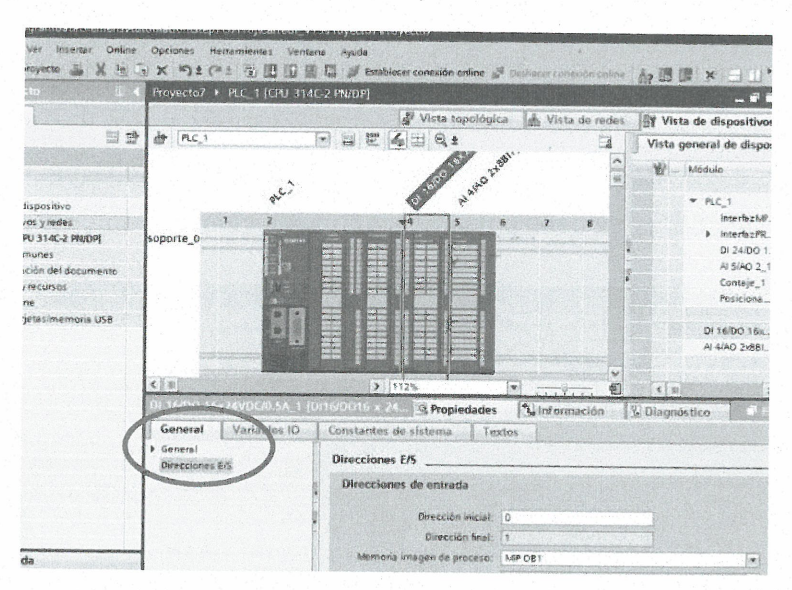


Figura 46

Y, por último, se puede ampliar la vista del dispositivo hasta que se visualicen las direcciones directamente en el mismo elemento o colocar el ratón para que muestre la dirección que es.

En la Figura 47 podemos comprobar esta situación.

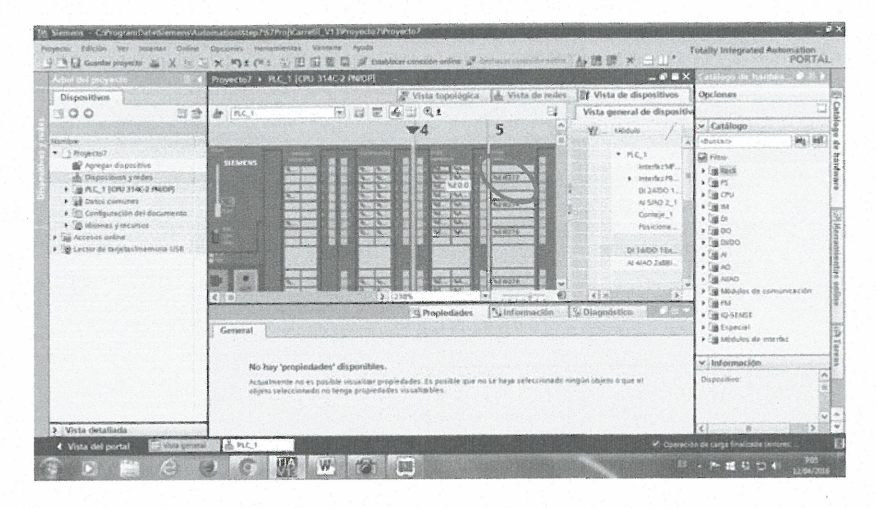


Figura 47

EDITOR DE PROGRAMAS EN STEP7 V5.5

Editor KOP

Ahora es el momento de escribir un programa y para eso se debe conocer el editor de programas del STEP 7. Se va a hacer una pequeña introducción en KOP, pero el objetivo será trabajar en AWL. Algunos ejercicios de KOP servirán para pasarlos posteriormente a AWL.

Para arrancar el editor se debe hacer desde el administrador. Se va desplegando el menú hasta obtener bloques. Pulsamos en bloques y en la parte derecha sale el bloque OB1, donde se escriben los programas (Figura 48):

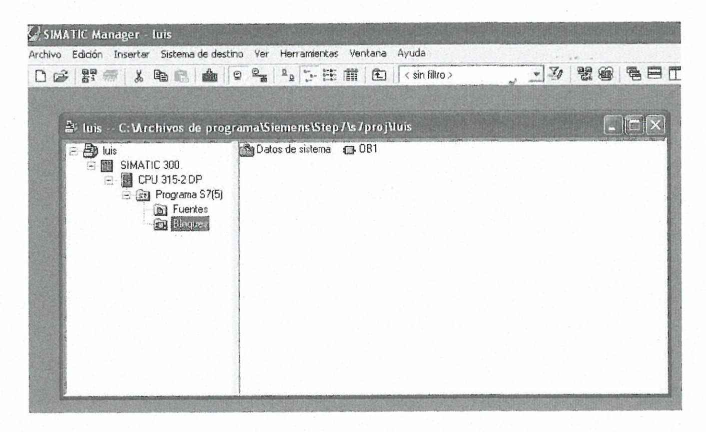


Figura 48

Si pulsamos dos veces sobre OB1, se abre y aparece una ventana nueva que es el editor, tal como se aprecia en esta Figura 49

KOP/AWL/FUP - [OB1 -- luis\SIMATIC 300\CPU 315-2 DP]
Archivo Edición Insertar Sistema de destino Test Ver Herramientas Ventana Ayuda
Contenido de: 'Entorno\Interface'
Interface
TEMP
Nombre
TEMP
OB1 : "Main Program Sweep (Cycle)"
Comentario:
Comentario:

Figura 49

Si no aparece igual que en la Figura 49, hay que ir a **VER** y activar **KOP**. El programa está dividido en segmentos y cada segmento está formado por una red eléctrica de programa. Las zonas en gris son para escribir comentarios sobre el programa y sobre cada segmento. Para **añadir más segmentos** hay que activar el icono que se marca en la Figura 50.

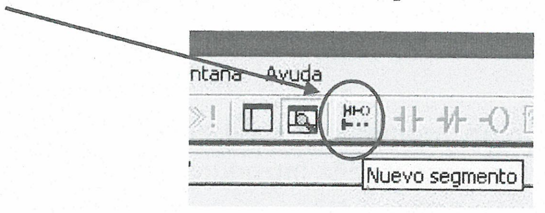
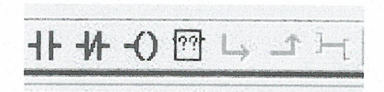


Figura 50

Se va a realizar el ejemplo anterior de dos contactos en serie que activan una salida. Se deben ir colocando los contactos, abiertos o cerrados, y construyendo el circuito con las líneas de dibujo que están en la parte superior del editor. Haremos igual con las salidas.



Según se van colocando los contactos y las salidas, podemos ir indicando las direcciones de cada contacto y salida. Nos tiene que quedar como en la Figura 51.

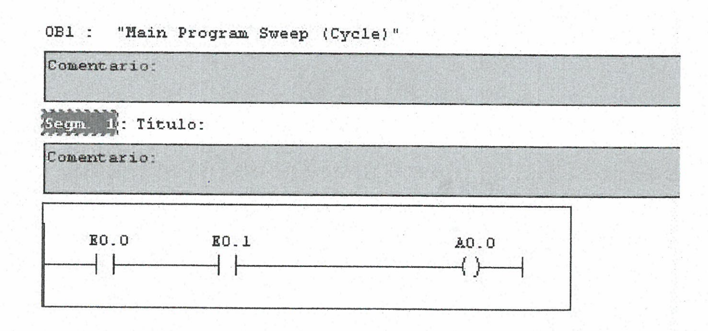


Figura 51

En el siguiente ejemplo se trata de poner dos contactos, uno abierto y otro cerrado, en paralelo. Para ello se deben utilizar las líneas de dibujo . Se activarán las líneas que se puedan utilizar en cada momento, colocando el cursor sobre la parte del circuito en la que se desee añadir alguna línea. En la Figura 52 se realiza paso a paso otro proceso. La mejor manera de aprender a utilizar este editor es practicando.

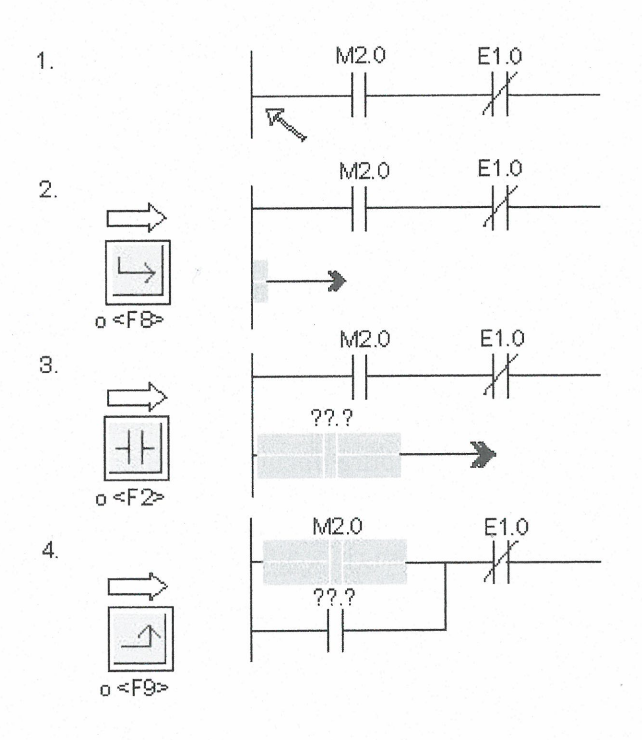


Figura 52

Es muy importante saber que en KOP cada rama eléctrica debe estar colocada, obligatoriamente, en segmentos diferentes. Esto significa que un caso como el de la Figura 53 no podrá realizarse en un mismo segmento. Serán necesarios dos segmentos, según se aprecia en la Figura 54.

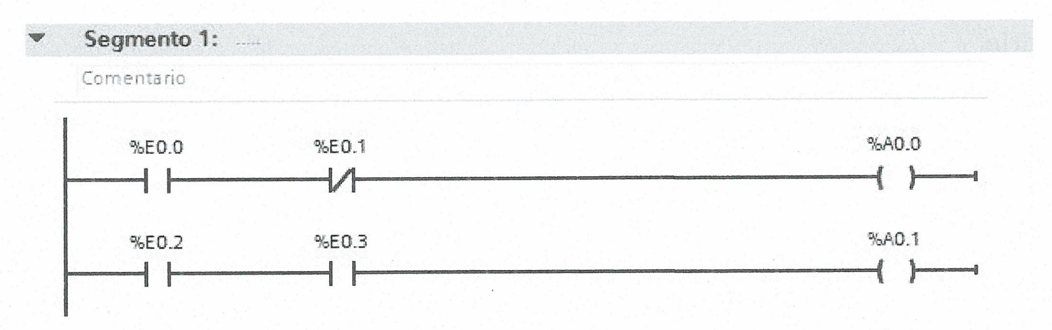


Figura 53

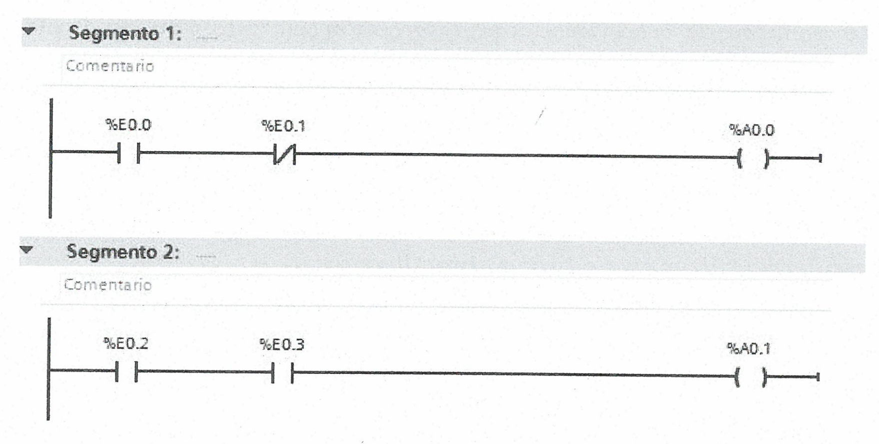


Figura 54

Editor AWL

De la misma manera que se ha hecho con KOP para seleccionar el editor de AWL, hay que abrir el menú desplegable ***VER*** y activar ***AWL***. El programa también está dividido en segmentos, si bien en AWL, no es obligatorio utilizarlos. Es decir, que se podría escribir todo el programa en el segmento 1. Como aspecto organizativo es más adecuado segmentar el programa, ya que de esta forma también se podrá documentar el programa por partes.

Este editor es mucho más sencillo, ya que, al ser texto, tan solo se deberá escribir el programa. Si en algún momento la instrucción se pone en rojo, quiere decir que no es correcta. Puede ser debido simplemente a no haber dejado un espacio entre instrucción y operando, o que la instrucción se ha escrito mal sintácticamente.

Hay que tener cuidado de no escribir el programa en la zona gris, que como hemos dicho es la destinada a texto, para realizar comentarios. Estos textos no se compilan y no se cargan en el PLC.

Existe una posibilidad de conversión de programas entre AWL y KOP, y viceversa. Para realizarlo, simplemente hay que ir a **VER** y activar el lenguaje al que lo queremos convertir. Es decir si se está en AWL y se desea convertir a KOP, una vez escrito el programa en AWL, se activa **VER/KOP**. La conversión no siempre es posible y esto no significa que el programa esté mal, solo quiere decir que, tal como lo hemos realizado, el programa no es capaz de convertirlo.

EDITOR DE PROGRAMAS EN TIA PORTAL

Editor KOP

El editor de KOP en TIA PORTAL es similar al del STEP7 V5.5. El OB1 se encuentra en el árbol del proyecto en la parte izquierda, bajo el epígrafe denominado **Bloques de programa**. Aquí ya aparece el programa Main(OB1). Para seleccionar el tipo de lenguaje se puede hacer con el botón derecho del ratón tocando en el nombre: Main(OB1). En la Figura 55 se muestra cómo hacerlo.

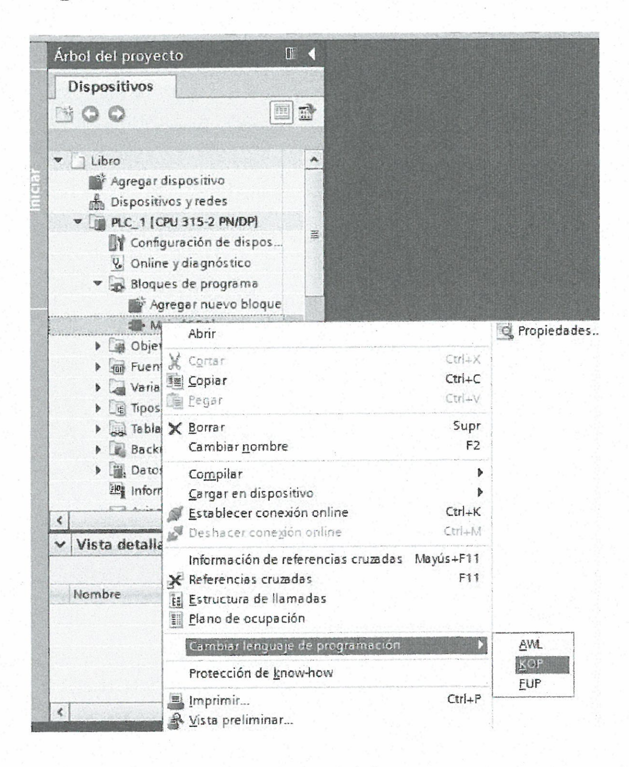


Figura 55

Una vez dentro del editor, la forma de ir añadiendo los contactos y salidas es similar al otro editor. En la Figura 56 se puede observar el aspecto de dicho editor.

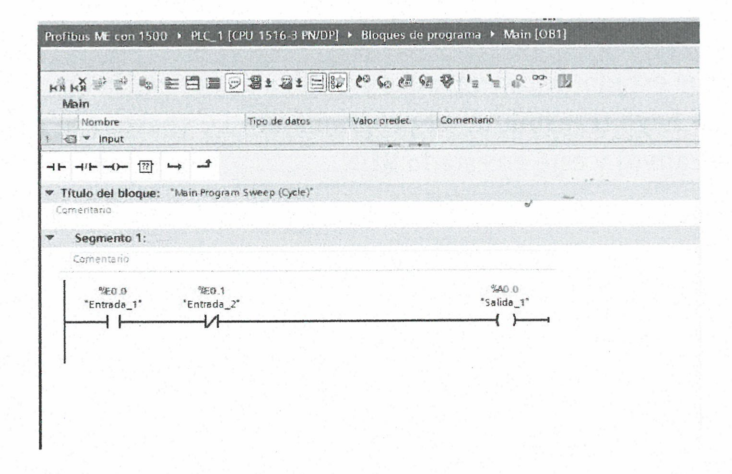


Figura 56

Editor AWL

Para entrar en el editor AWL se realiza la selección del programa como en el caso anterior. Tal como se ve en la Figura 55.

Una vez dentro, se van colocando las instrucciones, que se muestran en la parte derecha según se observa en la Figura 57.

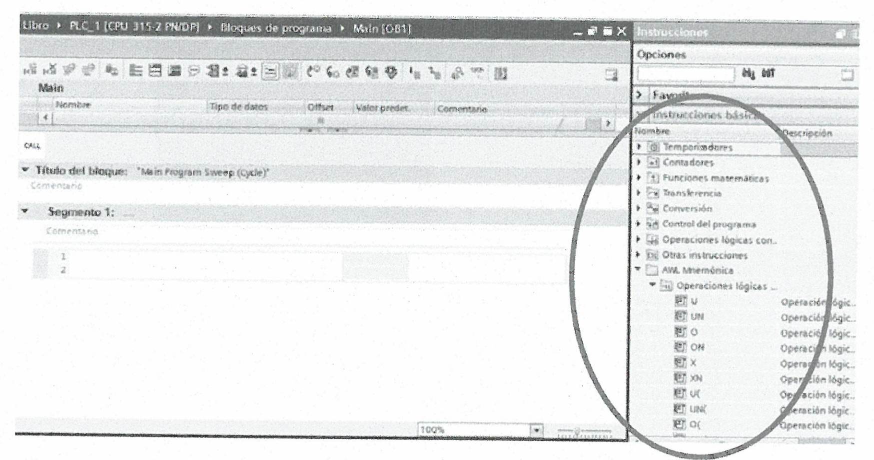


Figura 57

OBSERVAR E/S *ONLINE*

Tanto en KOP como en AWL se puede observar el comportamiento del programa en tiempo real, *online*. Para ello tenemos que activar, en la ventana del editor, el icono de las gafas:

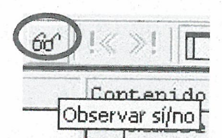


Figura 58

Para poder observar el estado de las entradas y salidas es necesario que, además de activar el icono de las gafas, se haya transmitido el programa al PLC (hasta ese momento el icono de gafas no aparece operativo) y esté en estado **RUN**.

Las imágenes en cada formato son como las de las Figuras 59 y 60.

Figura 59

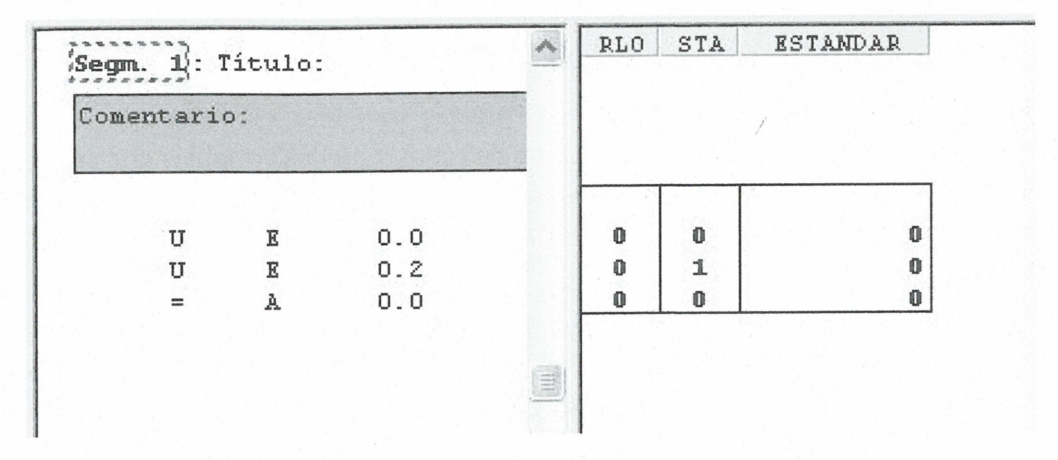


Figura 60

En el caso de AWL, vemos cómo visualiza el estado del RLO y de STA. Hay que diferenciar los dos bits. El RLO ya hemos dicho que es el bit que indica el estado lógico del programa. STA es el bit de estado de cada línea de programa. En el caso del ejemplo de la figura anterior, el STA indica que E0.0 no está activado y E0.2 sí lo está. Por eso el resultado del RLO en cada línea es 0, ya que son dos contactos en serie. Los valores ESTÁNDAR hacen referencia a otros valores, como temporizadores, contadores, etc.

Se pueden añadir más datos de información y también se pueden eliminar. Si hacemos clic con el cursor en esa zona, con el botón derecho se puede obtener más información, así como cambiar la notación, es decir, visualizar el dato en hexadecimal, decimal... En la Figura 61 se indican las distintas opciones.

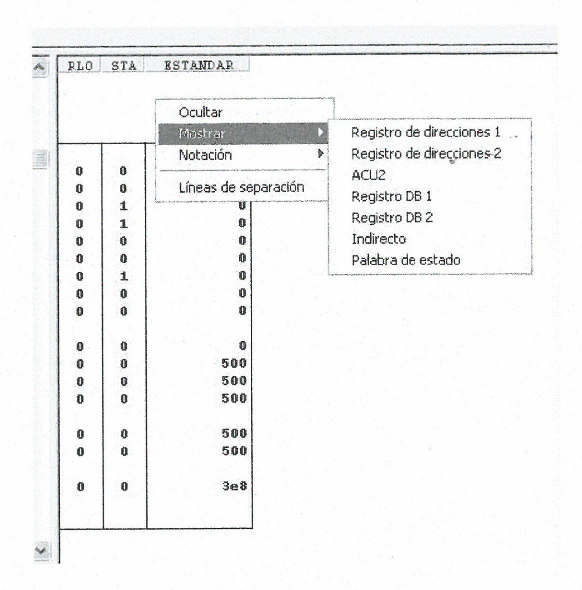


Figura 61

En TIA PORTAL ocurre lo mismo si se activan las gafas.

TABLA DE VARIABLES

Otra forma de poder observar el estado real de las variables es utilizando la **Tabla de variables**. Con ella se puede visualizar el estado de todas aquellas variables que se deseen, de cualquier tipo. Este método es más interesante que el anterior por muchas razones, entre otras porque en él se pueden disponer las variables en el orden que nos convenga y consultar únicamente las que nos interesen.

Para poder obtener la tabla de variables hay que ir a la pantalla de bloques del administrador, tal como se aprecia en la Figura 62, pulsar con el botón derecho del ratón sobre la ventana de bloques y seleccionar Insertar **nuevo objeto/Tabla de variables**:

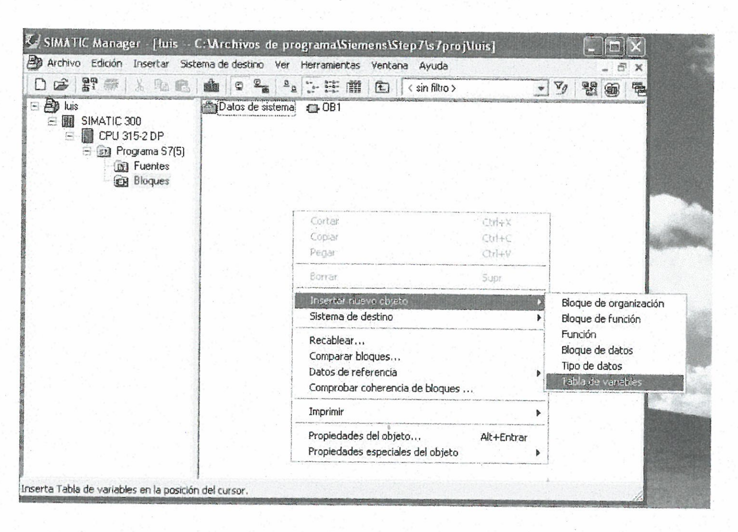


Figura 62

Al aceptar, aparece la pantalla en la que se puede asignar un nombre a la tabla de variables, si se desea (Figura 63). Dicha tabla aparecerá en la ventana de bloques (Figura 64). Desde esta ventana se pulsa dos veces sobre el nombre de la tabla y se abre.

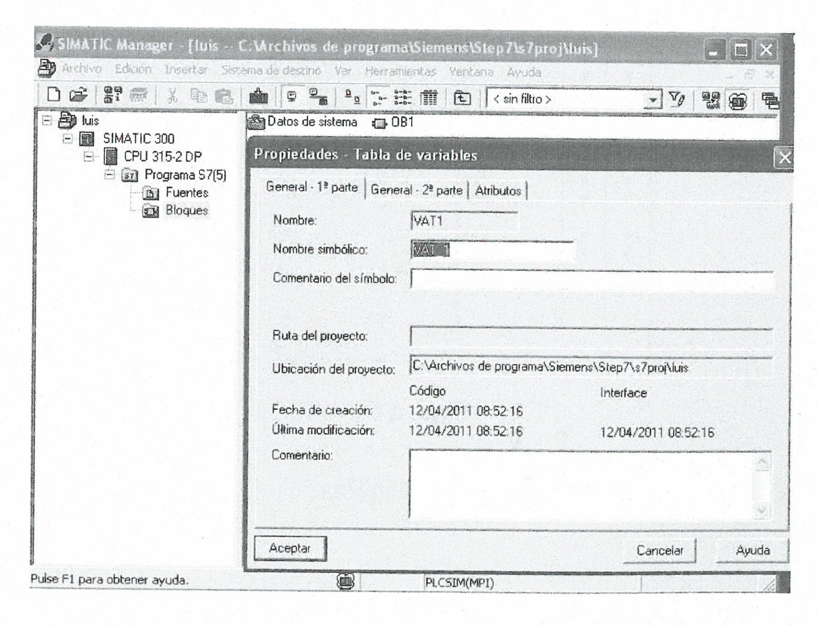


Figura 63

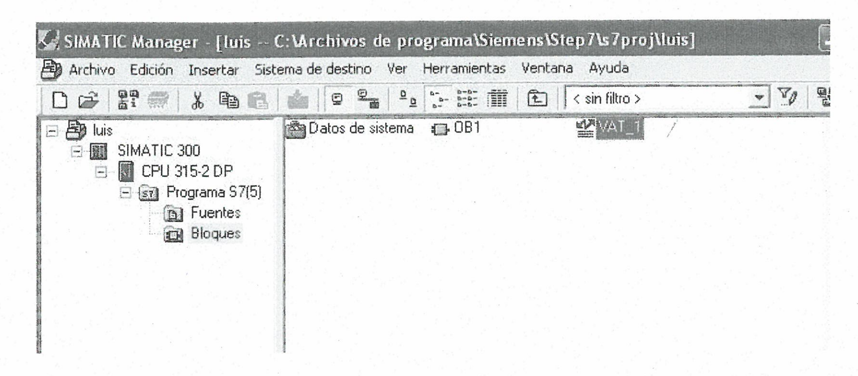


Figura 64

Cuando se abre la ventana de la tabla de variables, se escriben aquellas variables que se desean visualizar en Operando y se selecciona el formato de visualización. Una vez seleccionado el icono de las gafas, comienza a mostrarse su estado, tal como se aprecia en la Figura 65.

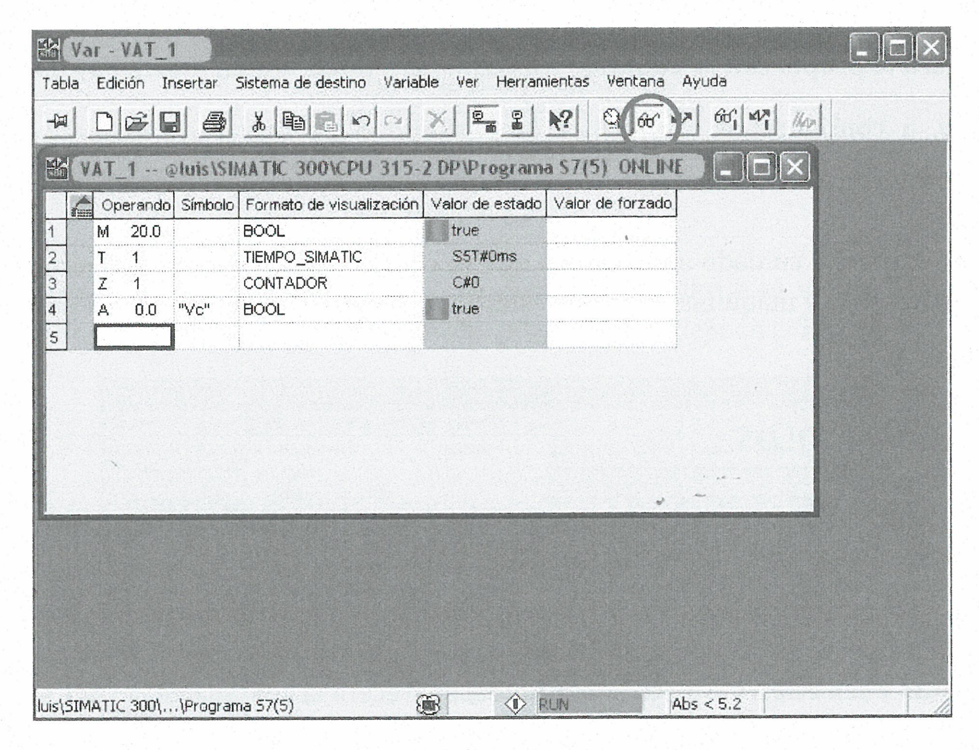


Figura 65

En TIA PORTAL hay que diferenciar entre la **Tabla de variables estándar**, que se encuentra en el apartado **Variables PLC**, y la **Tabla de** forzado **permanente**, que se sitúa en la **Tabla de observación y forzado permanente**. En las dos se puede consultar el estado de las variables.

La Figura 66 representa la tabla de variables que encontraremos en **Variables del PLC**.

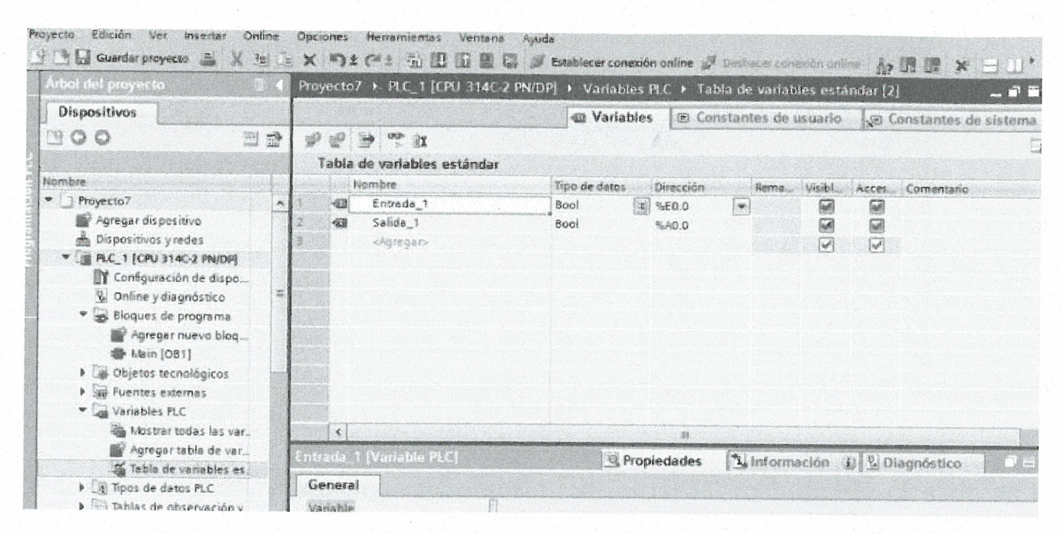


Figura 66

Si se desean ver las variables, se deberá abrir una **Tabla de observación;** y si se desea forzar, se hará lo propio con la **Tabla de forzado**. Para poder forzar, antes hay que activar el icono y, a continuación, . Existe la posibilidad de forzar de forma permanente, independientemente de la secuencia del programa, es decir, que se forzará siempre.

Hay que tener mucho cuidado al forzar, ya que se corre el **peligro** de ocasionar accidentes y daños a las personas y máquinas. Hay que estar muy seguro de lo que se desea hacer al forzar variables.

EDITOR DE SÍMBOLOS

Muchas veces es más interesante asignar a las entradas y salidas nombres que hagan referencia a la función que desempeñan que a su dirección.

Un símbolo es el nombre que se desea que tenga una entrada o salida. Para ello hay que editar dichos nombres. Se hace desde el editor de programas. Se debe desplegar el menú **Herramientas** y seleccionar **Tabla de símbolos**; aparecerá la ventana de la Figura 67.

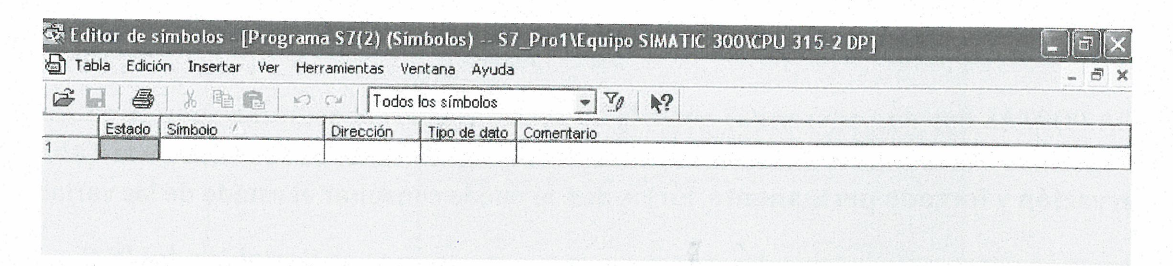


Figura 67

En **Símbolo** se debe especificar el nombre que se desea para la entrada o salida, mientras que en **Dirección** se indica la dirección física de la entrada o salida a la que se quiere asignar ese

nombre (símbolo). También se puede añadir un comentario sobre la función que desempeña la entrada o salida. En la Figura 68 se muestra un ejemplo.

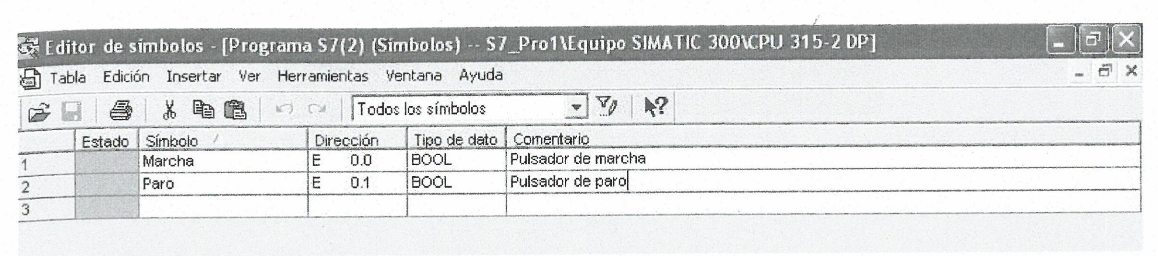

Editor de símbolos - [Programa S7(2) (Símbolos) -- S7_Pro1\Equipo SIMATIC 300\CPU 315-2 DP]

Tabla Edición Insertar Ver Herramientas Ventana Ayuda

	Estado	Símbolo	Dirección	Tipo de dato	Comentario
1		Marcha	E 0.0	BOOL	Pulsador de marcha
2		Paro	E 0.1	BOOL	Pulsador de paro
3					

Figura 68

Para que tengan efecto los símbolos, es necesario grabar la lista de símbolos. Ahora ya se puede nombrar a E0.0, por su dirección o directamente por su símbolo, **Marcha**. En el programa se alterna la visualización del símbolo o la dirección desde este **icono.**

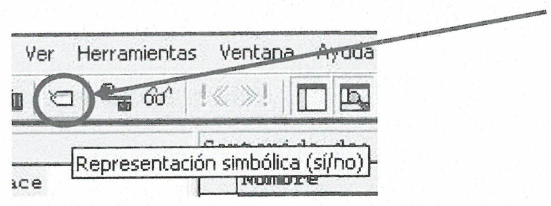


Figura 69

Si se desea que aparezcan las dos cosas, se puede hacer desde el menú **VER/Mostrar** y activando la opción **Información del símbolo**, tal como se refleja en la Figura 70.

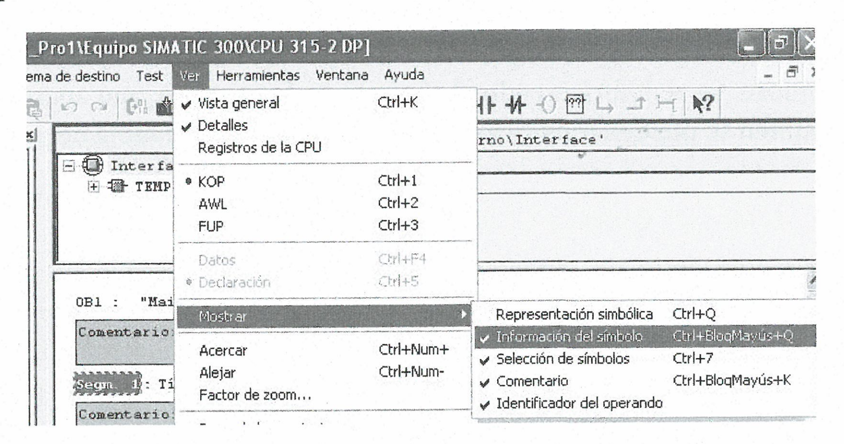
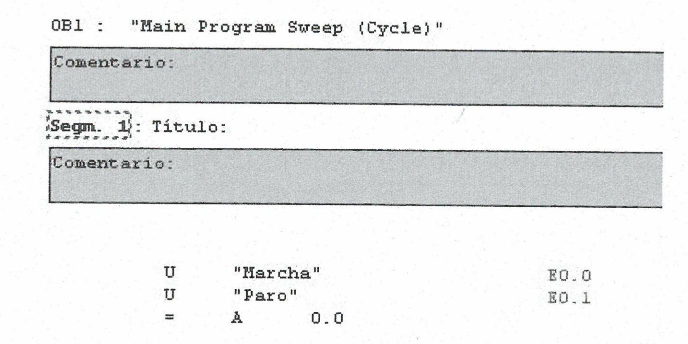


Figura 70

El programa con símbolos quedaría de la siguiente forma:



En TIA PORTAL el uso de símbolos viene impuesto por defecto, a cada variable le añade un símbolo denominado TAG y un número. Este nombre de símbolo se puede modificar *in situ* tocando en el nombre con el botón derecho del ratón. Se aprecia en la Figura 71.

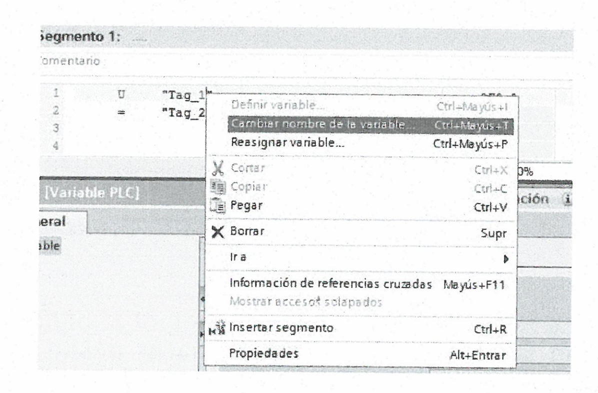


Figura 71

Al cambiar el nombre, ya se habrá creado el símbolo de la variable, que aparecerá en la tabla de variables. También se pueden modificar todas las variables en la tabla de variables.

EJERCICIOS

Se van a plantear una serie de ejercicios para practicar el editor KOP. Para realizar su comprobación, se deben cargar sobre el simulador y asegurarse de que responden al circuito eléctrico planteado. Esto se podrá hacer utilizando tanto la plataforma TIA PORTAL como STEP7 V5.5. Lo mejor será practicarlo en las dos. Igualmente se podrá programar sobre PLC S7-300 o S7-1500, ya que utilizan las mismas instrucciones.

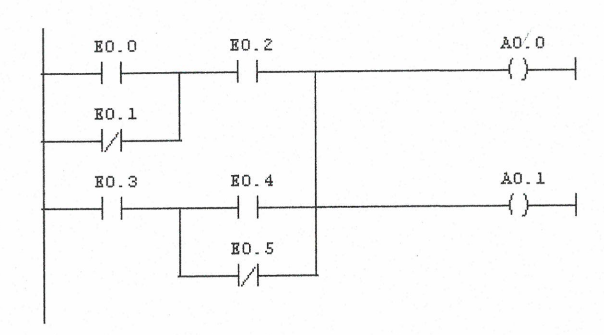
1- Pasar este programa en KOP al PLC y comprobar su correcto funcionamiento.

E0.0 E0.1 E0.2 E0.3 A0.0

E0.4 E0.5

E0.6 E0.7 E1.0

2- Pasar este programa en KOP al PLC y comprobar su correcto funcionamiento.



3- Escribe esta función lógica en KOP. Utiliza la tabla de símbolos con los siguientes nombres:

a === Marcha 1 (E0.0) b === Marcha 2 (E0.1) c === Acción Válvula 1 (E0.2)

d === Acción Válvula 2 (E0.3) S === Motor (E0.4)

$$S = (a \,.\, b) + (c \,.\, d) + a \,.\, \overline{b} \,.\, d + d(a + b)$$

4- Escribe esta función lógica en KOP:

$$S1 = ((a + b) \,.\, (c + \overline{a})) + d$$

$$S2 = (b \,.\, \overline{c} + a \,.\, c) \,.\, \overline{d}$$

Se pueden asignar a las entradas y salidas las siguientes direcciones:

a= E0.0 / b= E0.1 / c= E0.2 / d= E0.3 / S1=A0.0 / S2=A0.1

PROGRAMACIÓN EN AWL. Instrucciones de bit

En los siguientes apartados se seguirán utilizando las instrucciones de tipo bit, pero ya solo con el lenguaje AWL. Para comenzar, se partirá de un circuito en KOP y se tendrá que pasar a AWL. Inicialmente se hará sin usar paréntesis, aunque luego, cuando los programas sean más complejos, serán necesarios.

Para poder programar en AWL circuitos a nivel de bit solo se debe tener en cuenta cómo cambia el valor del RLO en cada una de las instrucciones. El siguiente circuito sirve de primer ejemplo:

E0.0 E0.2 A0.0
E0.1

Para realizar este circuito en AWL, se tiene que tener en cuenta el estado del RLO en cada paso de programa y comprobar que corresponde con el comportamiento del esquema eléctrico.

U E0.0	El estado de E0.0 se pone en serie con E0.2
U E.0.2	
O E0.1	El resultado de la serie anterior se pone en paralelo con E0.1
= A0.0	El valor del RLO se pasa a A0.0

Si este ejemplo se comenzara a hacer de otra forma, no sería correcto. Nos encontraríamos con el siguiente caso:

U E0.1
O E0.0
U E0.2
= A0.0

Es evidente que no es correcto, ya que el valor de E0.1 se pone en paralelo, pero solo con E0.0. El estado del RLO, formado por ese paralelo, se pone en serie con E0.2 y eso no es lo que indica el esquema.

Para afianzar esta idea realizaremos otro ejemplo.

E0.0 E0.2 E0.3 A0.0
E0.1

El programa correcto sería:

U E0.0
U E0.2
O E0.1
UN E0.3
= A0.0

Es evidente que con este procedimiento, sin más herramientas, en AWL se podrán realizar pocos circuitos. Es por ello que necesitamos utilizar los paréntesis.

Paréntesis

El uso de los paréntesis sirve para programar circuitos más complejos. Al abrir un paréntesis, se indica la operación que realizará.

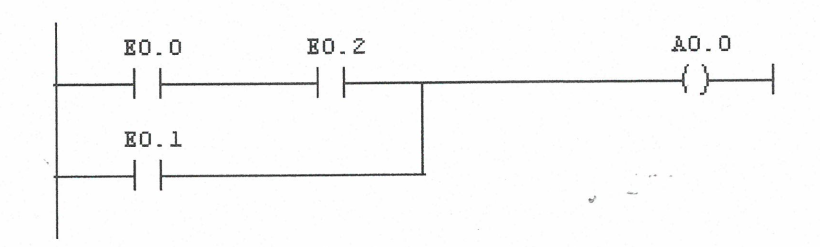
Instrucciones con paréntesis:

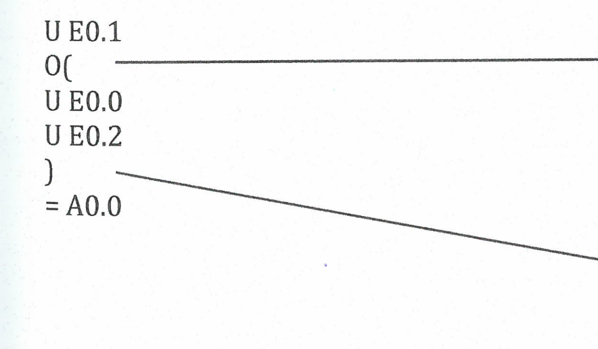
- **U(**
- **UN(**
- **O(**
- **ON(**

Cada una de esas instrucciones llevarán un cierre de paréntesis **)** .

Cuando se abre un paréntesis, el estado del RLO que hay justo antes del paréntesis se guarda en memoria. A partir de este punto el RLO es una primera consulta, es decir se reinicia. Cuando se cierra, hace la operación que indica el paréntesis con el RLO guardado en memoria.

Se va a realizar el ejemplo anterior pero con paréntesis.





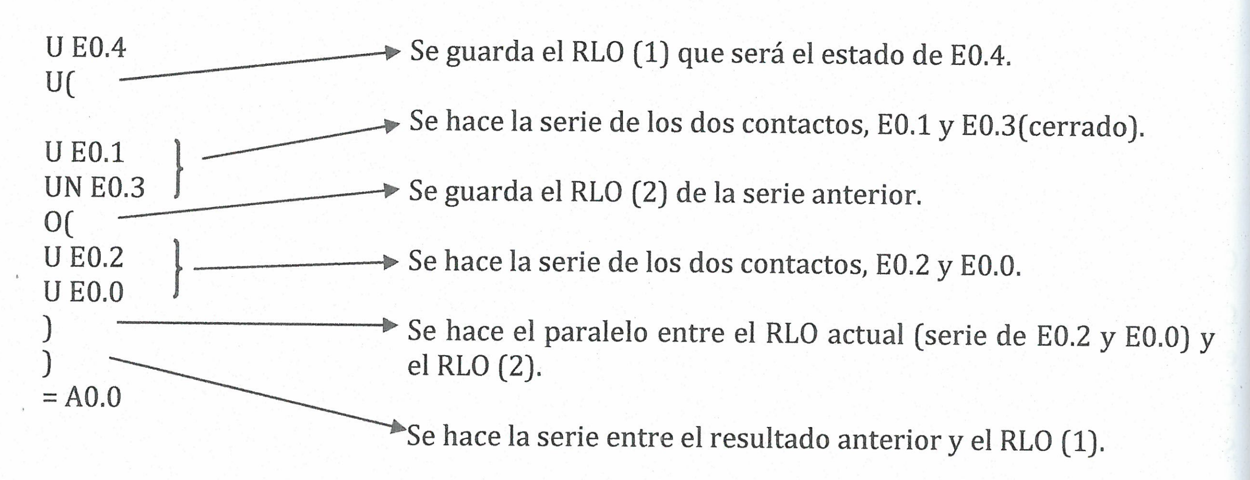
```
U E0.1
O(
U E0.0
U E0.2
)
= A0.0
```

O(→ Aquí el valor de RLO anterior se guarda en memoria y empezamos con un RLO nuevo, es el RLO en primera consulta que toma el valor de E0.0, como cuando se empieza un programa.

) → El resultado del RLO del paréntesis se pone en paralelo con el anterior RLO, guardado en memoria.

Otro ejemplo:

Se empieza a programar por el contacto E0.4. Es evidente que da igual por donde se empiece, y se hace por el contacto que va a generar el circuito más sencillo.



Se observa que se han utilizado dos paréntesis, uno dentro del otro. El principio siempre es el mismo: el RLO se guarda en una pila de memoria del autómata. Si dentro de un paréntesis hay otro, al abrir el segundo paréntesis, se guarda el RLO actual en la pila de memoria. Cuando se cierran los paréntesis, se van haciendo las operaciones en el orden lógico adecuado, es decir, que el último RLO guardado es el primero en salir y hacer las operaciones.

Vamos a ver otro ejemplo:

U E0.5

UN E

0.6

O E0.4

U(

U E0.1

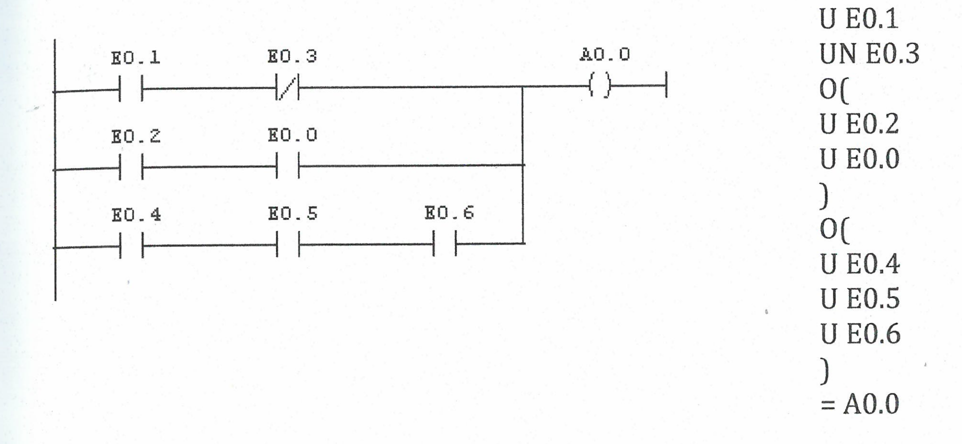
UN E0.3

O(

```
U E0.2
U E0.0
)
)
= A0.0
```

Todos estos programas pueden tener otras soluciones diferentes a las planteadas, dependiendo de cómo se siga el circuito. Para saber si es correcto el programa, hay que pasarlo al PLC o al simulador y comprobar que funciona según el comportamiento eléctrico del esquema.

Otro ejemplo:




```
U E0.1
UN E0.3
O(
U E0.2
U E0.0
)
O(
U E0.4
U E0.5
U E0.6
)
= A0.0
```

Este ejercicio tiene otra solución mucho más sencilla, sin tener que utilizar paréntesis. Cuando se trata de varias ramas en paralelo, se puede poner la instrucción *O* sin el paréntesis. Esto solo es válido para ramas en paralelo, no para ramas en serie.

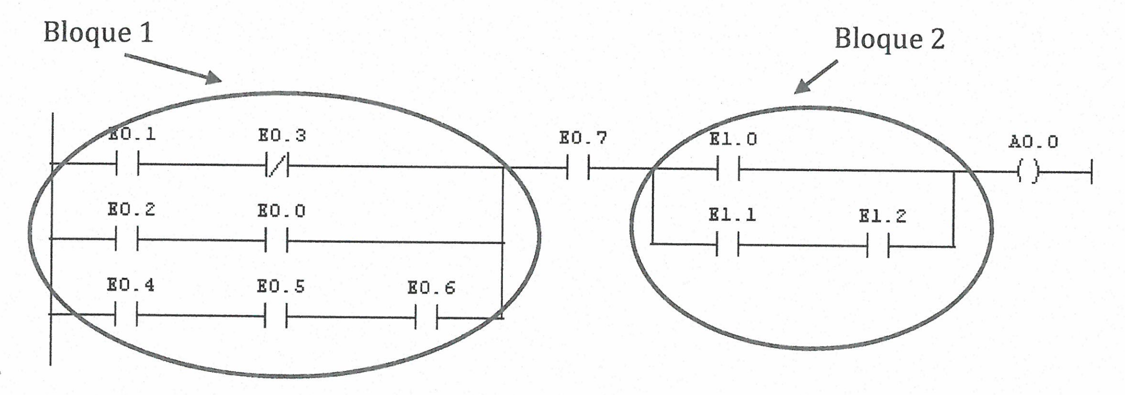
El ejercicio quedaría así:

```
U E0.1
UN E0.3
O
U E0.2
U E0.0
O
U E0.4
U E0.5
U E0.6
= A0.0
```

Uso de Marcas

En muchas ocasiones se pueden utilizar marcas de memoria para dar solución a circuitos complejos. Es otro método que puede resultar más sencillo que el de los paréntesis. Se trata de ir realizando partes equivalentes del circuito total, y a cada parte o equivalente asociarle una marca. Al final se deben realizar las operaciones lógicas con las marcas.

En el siguiente ejemplo podemos comprobar su aplicación. Se van a seleccionar dos bloques.



A cada bloque se le asocia una marca. Al bloque 1 la marca M0.0 y al bloque 2 la marca M0.1.

El circuito equivalente quedaría así:

M0.0 E0.7 M0.1 A0.0

Se realiza el programa de cada uno de los bloques. El bloque 1 es el mismo circuito que el ejercicio anterior, así que el programa será el mismo. Solo cambiará que la asignación se hace a la marca M0.0:

BLOQUE 1	BLOQUE 2	FINAL
U E0.1 UN E0.3 O(U E0.2 U E0.0) O(U E0.4 U E0.5 U E0.6) = M0.0	U E1.1 U E1.2 O E1.0 = M0.1	U M0.0 U E0.7 U M0.1 = A0.0

Ejercicios

1. Se pide escribir la siguiente función lógica en AWL:

$$S = a \cdot b + a \cdot c$$

2. Programar la siguiente función lógica en AWL:

$$S = a \cdot b + a \cdot b \cdot \overline{c} + b \cdot c$$

3. Pasar los siguientes circuitos a AWL:

a.

E0.1 E0.2 E0.3 E0.4 E0.5 A0.0
E0.6
E0.7 E1.0 E1.1
E1.2

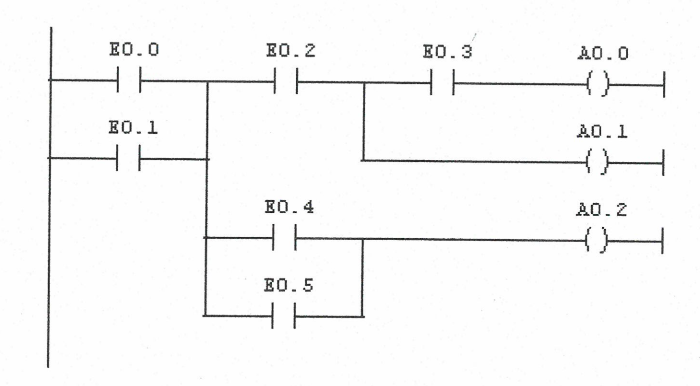
b.

E0.0 E0.2 E0.3 E0.4 A0.0
E0.6 E0.1 E0.5

c.

E0.0 E0.1 E0.2 E0.3 E0.7 E1.0 E1.1 E1.4 A0.0
E0.4 E0.5 E1.2 E1.3 E1.5
E0.6 E1.6

d.



Instrucciones set y reset

Las instrucciones de set y reset son órdenes de bit. Ponen el operando que se especifica en la instrucción a uno o a cero, respectivamente, siempre que el RLO sea UNO.

RESET (R): pone el operando a CERO.

SET (S): pone el operando a UNO.

Ejemplo:

```
U E0.0
S A0.0

U E0.1
R A0.0
```

En este ejemplo, la salida A0.0 se pone a UNO al activar E0.0. Permanece en este estado, aunque se suelte E0.0, hasta que se pulse E0.1, momento en el que la salida A0.0 pasa a CERO e igualmente permanece en este estado, aunque se desactive E0.1, hasta que se pulse de nuevo E0.0.

Este ejemplo sería la puesta en marcha de un motor de forma directa. E0.0 sería el pulsador de marcha y E0.1 el de paro (pero con contacto normalmente abierto en este caso).

Después de una orden S o R (al igual que pasaba con la asignación =), se reinicia el RLO.

Estas órdenes necesitan una activación, ya que dependen de que el estado del RLO sea UNO para que se ejecuten.

Hay que tener cuidado de no mezclar, en un mismo programa, asignaciones con S y R de una misma salida. Lo lógico sería que, si se ha utilizado la orden S o R, no se utilizara asignación.

En el siguiente programa se utiliza asignación de la salida A0.0 y S y R para la misma salida. Hay que escribirlo en el editor y pasarlo al PLC o simulador, y comprobar cómo no funciona correctamente la función S y R:

```
U E0.0
= A0.0
U E0.1
S A0.0

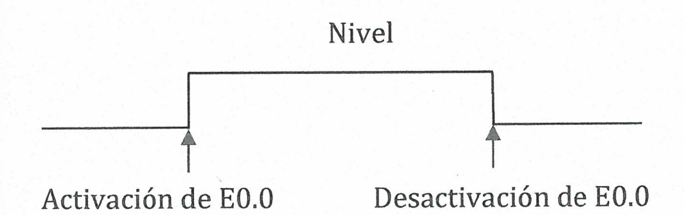
U E0.2
R A0.0
```

Operaciones con flancos

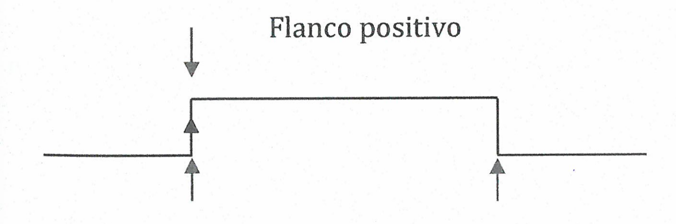
Cuando se habla de flanco, se está haciendo referencia al tiempo que transcurre entre el momento de la subida de una señal de 0 voltios a 1 (flanco de subida o positivo), o cuando pasa de 1 a 0 (flanco de bajada o negativo). Estos tiempos son muy reducidos. En el autómata se refleja en **un solo ciclo de ejecución**.

Si se concreta para la activación de un pulsador que activa una salida, se puede decir que, cuando la señal se activa por «nivel», es decir, como se ha hecho hasta ahora en todos los ejercicios, la salida se activa durante todo el tiempo en que permanece activado el pulsador. En este ejemplo pasa eso:

```
U E0.0
= A0.0
```



Sin embrago, si estuviera activado por flanco, solo el tiempo que trascurre desde que la señal pasa de 0 a 1, justo al accionar E0.0, estaría la señal activada y no durante el resto de tiempo que estuviera el pulsador accionado. Es evidente que ese tiempo es del orden de microsegundos o unos pocos milisegundos y, por lo tanto, el efecto que se apreciaría si se hiciera el ejemplo, sería nulo. El efecto que produce en el PLC este tipo de señal es que la salida se activa solo durante el tiempo de un ciclo de ejecución.



Las instrucciones para los flancos son:

Flanco positivo: FP M20.0 Flanco negativo: FN M20.1

Cada instrucción pueden llevar cualquier marca, pero debe ser exclusiva para esa instrucción. Es decir, **no deberá utilizarse en ningún lugar más**, ni para otro flanco, ni para otra aplicación en ese programa.

A continuación veremos un ejemplo sobre cómo se emplearían estas órdenes en un pulsador:

Para flanco positivo:

```
U E0.0
FP M20.0
S A0.0
```

Para flanco negativo:

```
U E0.0
FN
M20.1
S A0.0
```

Se va a comprobar en el simulador el efecto de esta orden en dos casos.

1º. En el siguiente programa se observa cómo la señal se activa al soltar el pulsador, ya que está activado por flanco negativo.

```
U E0.0
FN  M20.0
S A0.0

U E0.1
FP  M20.1
R  A0.0
```

Tanto para poner la salida a 1 como para ponerla a 0, se realiza en la **acción de desactivar** el pulsador. Se debe hacer despacio para poder comprobar el efecto.

2º. El siguiente programa se debe hacer primero sin flanco y luego con flanco. Hay que observar lo que sucede en cada caso. Si no se pone la instrucción de flanco, el valor de MW0 cuenta muchas veces con cada pulsación. Si se pone, se incrementa con cada activación. No hay que fijarse en las órdenes restantes, ya que no se han explicado todavía.

Para verlo en el PLC o el simulador se debe utilizar la marca 0 como palabra y ponerla en decimal para verlo mejor.

```
U   E    0.0
FP   M    20.0
SPB  l1

BEA
```

```
l1: L   1
    L   MW  0
    +I
    T   MW  0
```

Ejercicios

Hay que resolver los siguientes ejercicios de circuitos combinacionales con varias variables, sacando su tabla de verdad y simplificando por el método de Karnaugh. Escribir el programa en AWL.

1) Tres pulsadores ABC activan una salida S cuando: BC están pulsados - AC pulsados - AB pulsados - ABC pulsados.

2) Tres pulsadores ABC activan dos salidas, S y S1, cuando:

    1. B pulsado se activa S y S1

    2. A pulsado se activa S

    3. AC pulsados se activa S

    4. C pulsado se activa S1

3) Cuatro pulsadores ABCD activan una salida S cuando:

    1. C pulsado

    2. AD pulsados

    3. AB pulsados

    4. ABC pulsados

4) Tres pulsadores ABC activan cuatro salidas S0, S1, S2 y S3. Con A pulsado se activa S3. Con B pulsado se activan S0, S1 y S2. Con C pulsado se activa S0. Con BC pulsados se activa S2. Con AC pulsados S1 y S3. Con AB se activan S0 y S1. Con ABC pulsados se activa S0 y S2.

5) Diseñad un circuito combinacional que permita comparar 2 palabras, A y B, de dos bits cada una. Activar tres salidas de la siguiente forma:

    1. A=B.................... S0=1

    2. A>B.................... S1=1

    3. A<B.................... S2=1

6) El esquema de la Figura 72 muestra una alarma para automóvil empleada para detectar ciertas condiciones no deseables. Los tres interruptores se emplean para indicar el estado en que se encuentra la puerta del lado del conductor, el encendido y los faros. Realiza el programa en el autómata de manera que la alarma se active cuando se presenten cualquiera de las siguientes condiciones:

1. Los faros están encendidos mientras el coche está desactivado (motor parado).

2. La puerta está abierta mientras el coche está activado (motor en marcha).

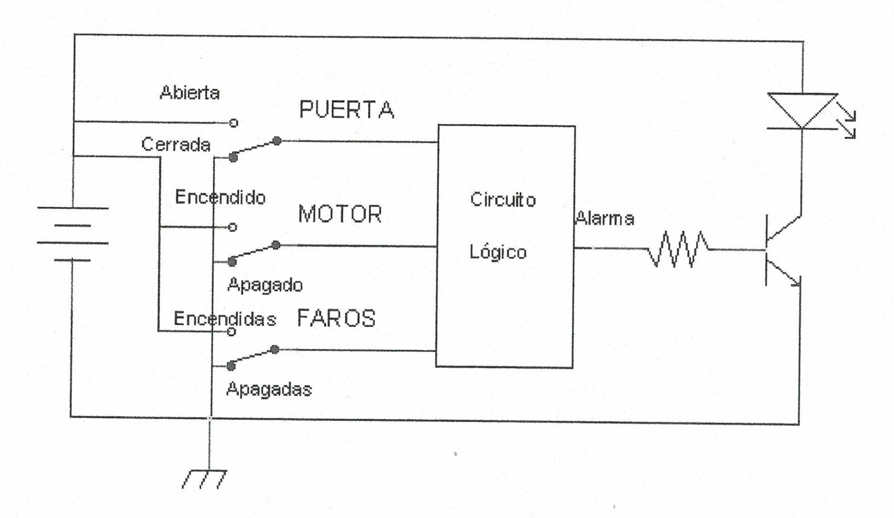


Figura 72

7) CONTROL DE UN GARAJE

Se trata de regular el acceso y la salida de un garaje que tiene un único acceso. Cuando un coche va a entrar, se enciende una luz roja en el interior del garaje. Cuando sale, se conecta otra luz naranja en el exterior. Otra lámpara naranja se instalará en el interior para informar a otros de que un coche está saliendo. Si un coche quiere entrar y otro quiere salir simultáneamente, el que entra deberá apartarse para que salga el coche del interior.

Para que la puerta se abra, debe activarse un sensor de presencia de coche. Hay uno situado en la entrada y otro en la salida. Además, se deberá activar un pulsador llave que hay en cada lado de la puerta.

Se trata de hacer la tabla de la verdad, simplificar y realizar el programa comprobando en el simulador que todo es correcto según lo planificado.

6. INSTRUCCIONES DE CARGA Y TRANSFERENCIA, ARITMÉTICAS, COMPARACIÓN Y SALTOS

INTRODUCCIÓN

En este tema se van a tratar algunas instrucciones que trabajan con el acumulador. Ya no solo se van a utilizar instrucciones a nivel de bit, ahora se trabajará con instrucciones de byte, palabra (W) y doble palabra (D). Estas órdenes trabajan con el acumulador.

INSTRUCCIONES DE CARGA Y TRANSFERENCIA

Cuando se desea realizar movimientos entre registros, el acumulador va a ser el registro «intermediario». Existe una orden que pasa el valor de un registro al acumulador 1 (ACU 1) y otra instrucción que hace lo contrario, pasa el contenido del acumulador 1 (ACU 1) a un registro. El registro será una marca o una E/S. No se pueden pasar datos directamente de un registro a otro.

ORDEN CARGAR (**L**): ejemplo: L MB 0.............. La marca de byte 0 pasa al acumulador

El ACU1 después de la carga queda de la siguiente forma:

ACU 1 H H | ACU 1 L L

			MB0

Otro ejemplo L MW0.......... La marca de palabra 0 pasa al acumulador.

ACU 1 H H | ACU 1 L L

		MB0	MB1

Otro ejemplo L MD0........... La marca de doble palabra 0 pasa al acumulador.

ACU 1 H H | ACU 1 L L

MB0	MB1	MB2	MB3

ACU 1 L L es el byte de menor peso del acumulador 1 y ACU 1 H H es el byte alto. Hay que fijarse dónde se colocan los bytes de las marcas en el acumulador. Esto, a veces, acarrea alguna confusión.

ORDEN TRANSFERENCIA (**T**): ejemplo: T MB 0............. El byte más bajo del acumulador 1 pasa a la marca de byte 0.

En las instrucciones de carga (L), cuando se carga, es decir, se introduce un dato en el acumulador 1 (ACU1), el valor que tuviera ACU1 pasa a ACU2 y el valor de ACU2 se pierde.

NOTACIÓN DE DATOS SIMPLES

En el tema *Tipo de datos y variables* se vio una tabla en la que aparecía la notación de los datos simples. Se van a incluir ahora algunos ejemplos en los que se muestra dicha notación.

Cargar números en binario:

L 2#10101 Es el número 21 en decimal

Cargar números en hexadecimal:

- El número es un byte (B): L B#16#2F
- El número es una palabra (W): L W#16#FF5A
- El número es una doble palabra (D): L DW#16#FF3FAB3F
- El número es BCD (C): LC Z1
- El número es un número decimal entero: L 310
- El número es un decimal doble entero: se coloca la L# al número, indicando que es de precisión doble entero: L L#310
- El número es un real: no se coloca coma, sino punto. Aunque no tenga decimales, si se trabaja con números reales, debemos poner el punto: L 23.45

L 23.0 (obligatorio poner el .0 si es un real)

INSTRUCCIONES DE COMPARACIÓN

Las instrucciones de comparación se utilizan para saber cómo son dos números entre ellos. Los números tienen que ser del mismo tipo. Y pueden ser:

- enteros,
- enteros dobles,
- reales.

Los números deben ser cargados en los acumuladores (ACU 1 y ACU 2), ya que son los que se utilizan en las instrucciones para comparar. El resultado de la comparación siempre es un bit.

Si se cumple el resultado de la comparación, el RLO se pone a uno (1), y, si no se cumple, se pone a cero. (0).

Las operaciones de comparación modifican el estado de los bits A1 y A0 del registro de estado. Todas las órdenes de comparación son **incondicionales.**

Para comparar, lo primero que se debe hacer es cargar un primer operando sobre el ACU1 y después otro sobre el ACU2. En realidad solo se tiene acceso a la carga sobre el ACU1, pero siempre el contenido del ACU1 pasa al ACU2 cuando se realiza una carga sobre ACU1. Es decir, que si se realizan dos cargas seguidas sobre el ACU1, estamos cargando el ACU1 y el ACU2. En el siguiente ejemplo se aclara esta situación:

```
L 30
L 23
```

En este caso, la primera orden carga el valor 30 sobre el ACU1. Después se vuelve a cargar el valor 23 sobre ACU1 ¿qué le pasa al número 30 que ha sido cargado en ACU1 antes? Ese valor, el 30, pasa al ACU2. La situación final es que:

ACU1= 23
ACU2= 30

Se puede resumir diciendo que el primer valor que se carga se quedará en ACU2 y el último en ACU1. Si se hicieran tres cargas seguidas, el primer valor introducido se pierde, ya que lo que hay en ACU2 se pierde al realizar otra carga sobre él.

COMPARACIÓN DE NÚMEROS ENTEROS Y DOBLES ENTEROS

Los números enteros son números de 16 bits. El indicador de número entero es **I.** En la siguiente tabla se indican las posibles comparaciones entre los dos acumuladores. Se ha colocado a la izquierda ACU2 y a la derecha ACU1 para que sirva de referencia. Se pueden realizar las siguientes comparaciones:

	INSTRUCCIÓN		SIGNIFICADO
ACU2	==I	ACU1	ACU2 igual a ACU1
ACU2	<>I	ACU1	ACU2 distinto de ACU1
ACU2	>I	ACU1	ACU2 mayor que ACU1
ACU2	<I	ACU1	ACU2 menor que ACU1
ACU2	>=I	ACU1	ACU2 mayor o igual que ACU1
ACU2	<=I	ACU1	ACU2 menor o igual que ACU1

Respecto a los números enteros dobles, es lo mismo. Cambia el rango del número que es de 32 bits. El identificador de número entero doble es **D**. La tabla es:

	INSTRUCCIÓN		**SIGNIFICADO**
ACU2	**==D**	ACU1	ACU2 igual a ACU1
ACU2	**<>D**	ACU1	ACU2 distinto de ACU1
ACU2	**>D**	ACU1	ACU2 mayor que ACU1
ACU2	**<D**	ACU1	ACU2 menor que ACU1
ACU2	**>=D**	ACU1	ACU2 mayor o igual que ACU1
ACU2	**<=D**	ACU1	ACU2 menor o igual que ACU1

COMPARACIÓN DE NÚMEROS REALES

Los números reales ocupan 32 bits y se puede decir lo mismo que se ha dicho para los números enteros. La única diferencia es que ahora son reales. En este caso hay que recordar que, para la parte fraccionaria, no se pone coma, sino punto.

	INSTRUCCIÓN		**SIGNIFICADO**
ACU2	**==R**	ACU1	ACU2 igual a ACU1
ACU2	**<>R**	ACU1	ACU2 distinto de ACU1
ACU2	**>R**	ACU1	ACU2 mayor que ACU1
ACU2	**<R**	ACU1	ACU2 menor que ACU1
ACU2	**>=R**	ACU1	ACU2 mayor o igual que ACU1
ACU2	**<=R**	ACU1	ACU2 menor o igual que ACU1

EJERCICIOS

1) Se trata de comparar los valores de las dos entradas digitales, AB0 y AB1. Se realizará el programa que cumpla con la siguiente tabla:

EB0 < EB1	A1.1 activada
EB0 > EB1	A1.2 activada

EB0 = EB1	A1.3 activada
EB0 = 0000 1111	A0.1 activada
EB1= 1111 0000	A0.2 activada
EB0 = 0111 y EB1= 1110	A0.3 activada

```
L   EB   0
L   EB   1
<I
=   A 1.0

L   EB   0
L   EB   1
>I
U   M   20.5
=   A   1.1

L   EB   0
L   EB   1
==I
U   M   20.3
=   A   1.2
```

En los tres primeros casos la asignación es sobre el byte AB1.

```
L   EB   0
L   2#1111
==I
=   A   0.1

L   EB   1
L   2#11110000
==I
=   A   0.2
```

Si EB0= 0000 1111, se activa la salida A0.1
Si EB0= 0000 0111, se activa la salida A0.2

INSTRUCCIONES ARITMÉTICAS

Las operaciones aritméticas que se pueden realizar son la suma, resta, multiplicación y división. Se pueden realizar sobre números enteros, dobles enteros y reales. Los dos acumuladores participan en las operaciones aritméticas, son el lugar donde se dejan los operandos y también donde se depositan los resultados. Esto debe tenerse en cuenta si se desean conservar los operandos, ya que estos, una vez realizada la operación, desaparecerán del acumulador 1 e incluso del acumulador 2, en algún caso.

Todas las operaciones **son incondicionales**, es decir, no dependen del valor del RLO, ni de otro factor para poder ejecutarse.

Operaciones aritméticas con enteros y enteros dobles

ENTEROS

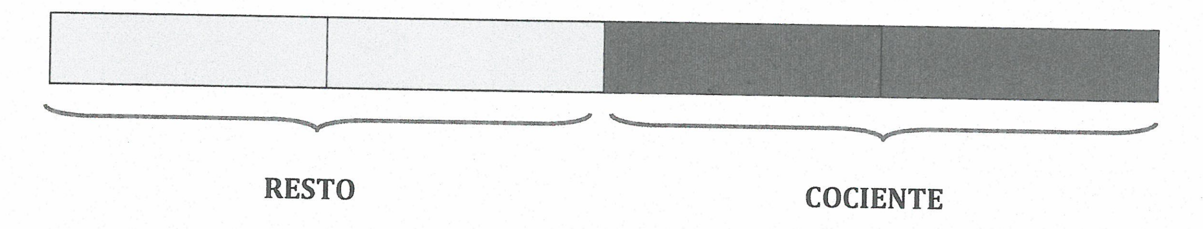
Las operaciones sobre números enteros, y su correspondiente instrucción, se indican en la siguiente tabla:

OPERACIÓN	INSTRUCCIÓN	SIGNIFICADO
SUMA	+I	ACU2 + ACU1 ⟹ ACU1
RESTA	-I	ACU2 - ACU1 ⟹ ACU1
MULTIPLICACIÓN	*I	ACU2 * ACU1 ⟹ ACU1
DIVISIÓN	/I	ACU2 / ACU1 ⟹ ACU1

Como son números enteros, solo se utilizará la palabra baja de los acumuladores, tanto en el caso de los operandos como en el resultado. El operando que hubiera en ACU1 antes de realizar la operación desaparece una vez ejecutada, ya que el resultado se almacena también en ACU1. El ACU2 permanece inalterado.

El caso de la división es una excepción. Como el resultado de la división tiene cociente y resto, se debe saber cómo deja esos resultados en el ACU1. En la palabra baja de ACU1 deja el cociente y el resto lo deja en la palabra alta de ACU1.

ACU 1:



RESTO **COCIENTE**

En la siguiente tabla hay un ejemplo para cada una de las operaciones, en el que se indica el resultado en el acumulador 1; también se indica cómo queda el acumulador 2 después de realizar la operación.

SUMA		RESTA		MULTIPLICACIÓN		DIVISIÓN	
L 8 L 2 +I	ACU1= 10 ACU2= 8	L 8 L 2 -I	ACU1= 6 ACU2= 8	L 8 L 2 *I	ACU1= 10 ACU2= 8	L 8 L 2 /I	ACU1L = 4 ACU1H = 0 ACU2= 8

DOBLES ENTEROS

Las operaciones sobre números enteros dobles, y su correspondiente instrucción, se refleja en la siguiente tabla:

OPERACIÓN	INSTRUCCIÓN	SIGNIFICADO
SUMA	+D	ACU2 + ACU1 ⟹ ACU1
RESTA	-D	ACU2 - ACU1 ⟹ ACU1
MULTIPLICACIÓN	*D	ACU2 * ACU1 ⟹ ACU1
DIVISIÓN	/D	ACU2 / ACU1 ⟹ ACU1

Como son enteros dobles, ocuparán todo el acumulador, tanto en los operandos como en los resultados. En el caso del resultado de la **división**, el cociente ocupa **todo el ACU1**. Para poder conocer el resto se debe ejecutar otra instrucción que solo calcula el resto de la división. Es la orden **MOD** que calcula el resto entre números enteros dobles, entre ACU2 y ACU1 (ACU2/ACU1). En los ejemplos veremos cómo hacerlo.

Hay que recordar que para introducir enteros dobles en el acumulador la orden es:

L L#entero doble

En la siguiente tabla se incluye un ejemplo para cada una de las operaciones, en el que se indica el resultado en el acumulador 1; también se indica cómo queda el acumulador 2 después de realizar la operación.

SUMA		RESTA		MULTIPLICACIÓN		DIVISIÓN	
L L#8 L L#2 +D	ACU1= 10 ACU2= 8	L L#8 L L#2 -D	ACU1= 6 ACU2= 8	L L#8 L L#2 *D	ACU1= 10 ACU2= 8	L L#8 L L#2 /D	ACU1 = 4 ACU2= 8

En el caso de la división, si se desea conocer el resto, se debe utilizar la orden MOD, que calcula el módulo de la división, es decir, el resto. A continuación se incluye un ejemplo.

L L#8 L L#2 /D	AC2 / AC1 ACU1= 4 Y ACU2= 8
T MD20 MOD T MD21	Guarda el cociente en la marca de doble palabra 20. Calcula el resto entre AC2 Y AC1 (ACU2/ACU1). Guarda el resto en la marca de doble palabra 21

Estos ejemplos anteriores se deben realizar para comprobar su funcionamiento. Debe utilizarse el simulador y visualizar los acumuladores.

Al final de este tema se realizarán más ejercicios con operaciones aritméticas.

INSTRUCCIONES DE SALTO

En este tema se han visto un grupo de instrucciones que son incondicionales, es decir, que no dependen de nada para ejecutarse. Para aquellas instrucciones que son incondicionales puede ser útil contar con un método para convertirlas en condicionales. Esta será una de las aplicaciones de los saltos.

Las órdenes de saltos modifican el orden secuencial normal de un programa llevando su ejecución a otro lugar. A ese otro lugar se le llama **META**. Los saltos pueden ser condicionales o incondicionales. Los incondicionales saltarán a la **meta** siempre, sin ninguna condición. Los condicionales irán a la **meta** solo si se cumple la condición, por ejemplo, que el RLO sea uno. De otra forma, seguirá la secuencia normal ejecutando la siguiente orden al salto.

SALTOS CONDICIONALES

Las instrucciones de salto condicional más utilizadas son las que dependen del estado del RLO. Hay otras órdenes condicionales que, aunque menos utilizadas, conviene al menos conocer, por lo que al final del apartado se incluirá una tabla con esas otras instrucciones de salto.

La siguiente tabla indica las órdenes condicionales al estado del RLO:

SPB meta	Salta si el RLO = 1
SPBN meta	Salta si el RLO = 0
SPBB meta	Salta si el RLO = 1 y el bit RB = 1
SPBNB meta	Salta si el bit RLO = 0 y el bit RB = 1
Donde «meta» es el nombre de la etiqueta que se debe colocar en la instrucción y en el destino del salto. La etiqueta puede ser cualquier conjunto de letras y números que concluya con dos puntos(:). No debe superar los 4 caracteres y debe empezar por un carácter letra. RB es el bit 8 del registro «Palabra de estado».	

INSTRUCCIÓN DE FIN DE MÓDULO (BEA)

La instrucción de fin de módulo se utiliza, fundamentalmente, en los saltos. Se supone que si se usa un salto, es porque **SOLO** se desea llegar allí cuando se salte con una de las órdenes de salto y no por otro camino.

Para evitar que los saltos se mezclen y se ejecuten sin haber ejecutado expresamente su orden de saltar, se debe poner BEA. En este caso el programa se termina y se vuelve a ejecutar un nuevo ciclo. Por ello, todo lo que escribamos debajo de BEA no se ejecutará. Es evidente que si se pone algo debajo de BEA, serán las metas de los saltos que se deseen ejecutar.

Ejemplo

En este ejemplo la salida A0.0 se activa con E0.2 o con E0.3, dependiendo de si E0.0 está abierto o cerrado.

```
U   E   0.0          Si se activa E0.0, el programa salta a L1.
SPB  l1

UN  E   0.0          Si no se activa E0.0, el programa salta a L2.
SPB  l2

BEA                  Si se ejecuta BEA, el programa no sigue con las siguientes órdenes,
                     vuelve a empezar.

l1: U   E   0.2      La salida A0.0 se activa con E0.2 y E0.0 activado.
=   A   0.0          Aquí el BEA impide que las ordenes que no pertenecen al salto de
BEA                  META l1: se puedan ejecutar.

l2: U   E   0.3
=   A   0.0          La salida A0.0 se activa con E0.3 y E0.0 NO activado
```

La siguiente tabla muestra otras instrucciones de salto:

En función de los bits de estado RB/OV/OS	
SPBI meta	Salta si RB = 1
SPBN meta	Salta si RB = 0
SPO meta	Salta si OV = 1
SPS meta	Salta si OS = 1
En función del estado del ACU1 después de realizar una operación	
SPZ meta	Salta si ACU1 = 0
SPN meta	Salta si ACU1 no es 0
SPP meta	Salta si ACU1 > 0
SPM meta	Salta si ACU1 < 0
SPMZ meta	Salta si ACU1 <= 0
SPPZ meta	Salta si ACU1 > =0
SPU meta	Si el resultado no es válido (por ejemplo, que en una operación con números reales uno de los operandos no sea real).

SALTOS INCONDICIONALES

Como saltos incondicionales existen dos instrucciones, SPA y SPL. SPA es el salto incondicional a una meta y SPL es el salto a una lista de metas.

La orden **SPA meta** salta siempre a la meta, sin ninguna condición. Las instrucciones que haya por debajo de SPA no se ejecutarán nunca, a no ser que se llegue a ellas desde otros saltos, que será lo normal. Para evitar casos como este, todas las metas se programan al final del mismo.

La orden **SPL meta** salta a una lista de saltos incondicionales. Saltará a un lugar u otro en función del valor del ACU1. Veremos un ejemplo para diferenciar el uso de SPA y de SPL.

En el siguiente ejemplo se carga sobre el ACU1 el estado del byte de entradas 0 (EB0). Si el valor de ACU1= 0, el programa salta incondicionalmente al primer salto de la lista de saltos incondicionales, en este caso a L1. Aquí al pulsar sobre E1.0, se activa la salida A1.0.

Si ACU1=1, saltará al segundo salto de la lista, es decir, a L2. Ahora E0.0 activa la salida A1.1.

Si ACU1=2, saltará al tercer salto de la lista, L3. Aquí ahora E0.0 activa A1.2.

Pero si ACU1 tiene un valor superior al número de saltos incondicionales disponibles en la lista (en este ejemplo, si ACU1 es superior a 2), se salta a la meta que tiene la instrucción SPL, es decir, en este caso a nada. En el ejemplo la acción es poner todos los bits de la salida de byte 0 (AB0) a uno.

```
L  EB0
SPL nada
SPA L1
SPA L2
SPA L3
```

La lista siempre tendrá esta distribución, la orden SPL.

```
nada:  L  B#16#FF
T  AB0
BEA

L1:     U E1.0
= A1.0
BEA

L2:     U E1.0
= A1.1
BEA

L3:     U E1.0
= A1.2
```

Las metas siempre tendrán esta distribución, a continuación de la lista de saltos.

Si en este programa se quitaran las instrucciones BEA, su funcionamiento sería otro totalmente distinto. El programa quedaría así:

```
L  EB0
SPL nada
SPA L1
```

```
          SPA L2
          SPA L3
nada:  L  B#16#FF
T  AB0

L1:     U E1.0
= A1.0

L2:     U E1.0
= A1.1

L3:     U E1.0
= A1.2
```

Hay que observar que, si se salta a L1, a partir de ahí se ejecutan todas las ordenes, incluidas las que tiene la meta L2 y L3. Lo mismo pasaría si salta a L2, que se ejecutarían también las órdenes de la meta L3. E igual si se salta a la meta «nada», que se ejecutarían todas las demás metas.

No es necesario incluir la instrucción BEA del final del programa, porque el ciclo de ejecución del autómata ya reinicia el ciclo después de la última orden.

EJERCICIOS

1. Con la entrada E0.5 se carga el valor de EB1 en el acumulador 1 (ACU 1). A cada pulso de la entrada E0.0 se multiplica ese valor por 2 hasta un máximo de 255. A cada pulso de E0.1 se divide el valor por 2 hasta un mínimo de 1.

2. El programa realiza la suma de las entradas EB0 y EB1. Si el valor de la suma es mayor de 100, se activa la salida A0.0 intermitente (t = 1 s). Si es menor o igual que 100, se activa la salida A0.1 intermitente (t = 2 s). Si las dos entradas (EB0 y EB1) son cero, se activa AB1 intermitente (t = 1 s).

3. El programa realiza operaciones aritméticas en función del valor de la entrada EB0, según la siguiente tabla:

EB0=0	5 * 20
EB0=1	20/2
EB0=2	100+155
EB0=3	200-72
EB0=4	100*2
EB0=5	2+2

El resultado lo saca por la salida AB0. Hay que utilizar tabla de saltos.

7. TEMPORIZADORES Y CONTADORES

INTRODUCCIÓN

El uso de temporizadores y contadores resulta fundamental para cualquier sistema automático. Cada PLC tendrá unos tipos de temporizadores y contadores, y una cantidad diferente.

Dentro de la serie Simatic S7 300 y 400, disponemos de cinco tipos distintos de temporizadores. La cantidad de temporizadores y contadores dependerá del PLC disponible. Por ejemplo, el PLC S7 315 2DP dispone de 128 temporizadores y 64 contadores.

Como ya se ha dicho en otras secciones de este libro, estos contadores de la familia S7 300/400 se pueden programar de la misma manera en STEP7 V5.5 y TIA PORTAL. También se podrán utilizar el PLC S7-1500. Este autómata dispone además de otros temporizadores. Al final del libro se hablará de los contadores IEC del PLC 1500.

Los temporizadores, en el nemónico alemán, se identifican con la letra T y los contadores con la Z.

TEMPORIZADORES

El proceso de un temporizador es paralelo al de la CPU, es decir, que son procesos independientes. Una vez arrancado el temporizador, la CPU no emplea tiempo ni dedicación en ese temporizador.

Para utilizar un temporizador es necesario realizar tres pasos:

1. Cargar el tiempo de temporización.
2. Arrancarlo con un tipo de funcionamiento.
3. Utilizarlo, es decir, consultar su estado con operaciones de bit.

Un temporizador también se puede borrar, es decir, parar su temporización. Cuando se carga un tiempo en un temporizador, este tiempo se almacena en su correspondiente registro, que es de 16 bits. Cuando se arranca, ese registro se va decrementando hasta llegar a cero, que será el final del tiempo.

USO DE LOS TEMPORIZADORES

Hay cinco tipos diferentes de temporización en los PLC S7 300:

I. Modo Impulso (**I**)

II. Modo Impulso prolongado (**V**)

III. Modo Retardo a la conexión (**E**)

IV. Modo Retardo a la conexión con memoria (**S**)

V. Modo a la desconexión (**A**)

I V E S A

Se van a estudiar cada uno de ellos. Se realizará su estudio sobre la práctica, pero antes hay que analizar cómo se programan. Según se ha visto en el apartado anterior hay tres pasos a cumplir en el uso de los temporizadores.

1. **Cargar el tiempo**. Se hace mediante la constante de tiempo Simatic, que es una forma de estandarizar el formato de tiempo para los autómatas de SIEMENS. Esa constante de tiempo se indica en el formato SIMATIC del siguiente modo: S5T#tiempo, donde tiempo es el valor en horas, minutos, segundos y milisegundos. El valor máximo de tiempo es 2 horas 46 minutos 30 segundos y 00 ms o, lo que es lo mismo, 9990 segundos.

Ejemplos para cargar diferentes tiempos:

L S5T#3s, L S5T#2h2m3s, L S5T#95ms, L S5T#4h20m

Cuando se carga un valor de tiempo, Step 7 escoge una base de tiempos con la que va decrementando el registro. Esta base de tiempos la elige automáticamente el propio programa optimizando a la más adecuada en función del tiempo a temporizar. Las bases de tiempo son cuatro:

10 ms, 100 ms, 1 s y 10 s.

Hay otra forma de cargar el tiempo en el temporizador. En este método se debe seleccionar la base de tiempos y cargar el valor del temporizador en BCD. La sintaxis es **L W#16#abcd**, donde **a** es el código de la base de tiempos y **bcd** es el valor de temporización en BCD.

El código de la base de tiempos es el siguiente:

10 ms = 0 100 ms = 1 1 s = 2 10 s = 3

Por ejemplo, si se desea cargar un tiempo de 20 segundos, la orden sería:

L W#16#2020

También se puede consultar o guardar el valor de tiempo de un temporizador mediante la orden LC T1, donde **T1** puede ser cualquier n.º de temporizador. Igualmente, puede cargarse dicho tiempo en otro temporizador si se desea. Sería así:

```
LC T1
T  T2
```

Carga el valor de T1 en la parte baja de ACU 1, expresado en BCD, en el acumulador 1 y transfiere el valor del acumulador 1 (T1) a T2.

Si se quiere guardar en binario, sería:

```
L T1
```

Carga el valor de T1 en la parte baja de ACU 1, expresado en binario, en el acumulador 1.

2. **Arrancar el temporizador** con un modo de trabajo. Se utiliza la orden ***Stipo*** **T1**, donde **tipo** es el modo de trabajo y **T1** puede ser cualquier temporizador. Esta instrucción necesita una activación. Depende del valor del RLO: si es uno arrancará el temporizador.

Este sería un ejemplo para el modo Impulso (**I**): El temporizador T1 arrancará en modo impulso cuando se active E0.0.

```
U E0.0
SI T1
```

3. **Utilizar el temporizador.** Se puede saber si se ha terminado la temporización consultando el estado del **bit** del temporizador. En función del tipo de temporizador ese bit se activará de una u otra forma. En el siguiente punto se comprobarán los distintos modos de trabajo y, por lo tanto, cómo varía ese bit. Las instrucciones para su utilización serían:

```
U T1
=  A0.0
```

Donde la salida A0.0 se actualizará siguiendo el comportamiento de T1, que dependerá de su modo de trabajo.

MODOS DE TRABAJO DE LOS TEMPORIZADORES

En el apartado anterior se han indicado cinco tipos diferentes de comportamiento de los temporizadores y cómo poder utilizarlos.

Ahora se va a comprobar cómo actúan esos temporizadores según los diferentes modos de trabajo. Para ello se realizarán los correspondientes programas de cada uno y se rellenará la siguiente tabla. En ella se destacan las diferencias fundamentales.

Para realizar estas comprobaciones es necesario incluir en el OB1 los diferentes programas que se indican en la tabla para cada tipo. No se deben escribir todos a la vez en el OB1, ya que se utiliza el mismo temporizador. Hay que incluir uno cada vez y comprobar.

En el simulador se puede utilizar la entrada E0, la salida A0 y el temporizador T1, donde aparecerá el tiempo y su base de tiempos. En la Figura 73 se observa cómo debe estar el simulador.

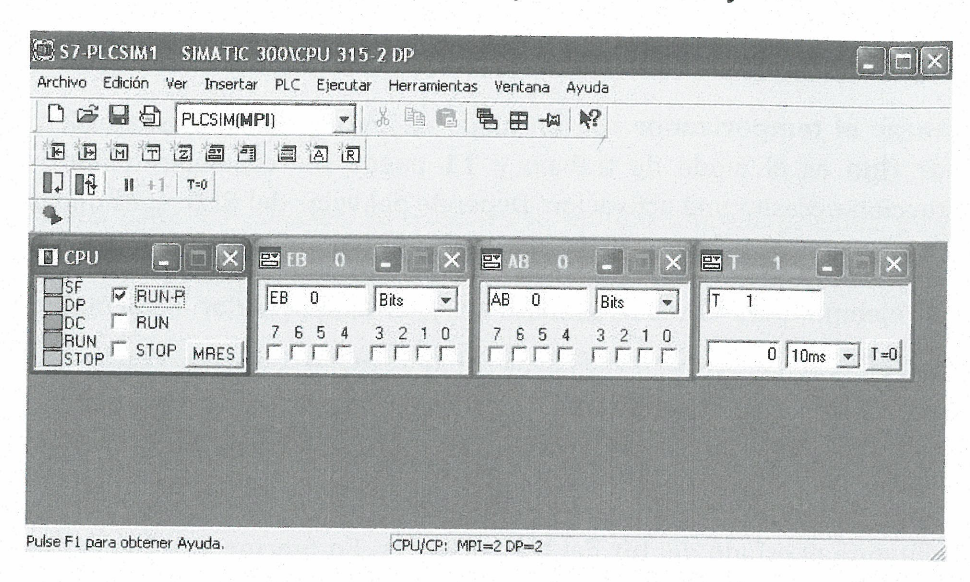


Figura 73

Seguidamente hay dos tablas, una está vacía y hay que completarla según los resultados que se obtengan; la otra está ya completada y hay que compararla con la que se ha creado.

TIPO	I	V	E	S	A
PROGRAMA	L S5T#8S U E0.0 SI T1 U T1 = A0.0	L 5T#8S U E0.0 SV T1 U T1 = A0.0	L S5T#8S U E0.0 SE T1 U T1 = A0.0	L S5T#8S U E0.0 SS T1 U T1 = A0.0	L S5T#8S U E0.0 SIA T1 U T1 = A0.0
¿Cómo está A0.0 (bit T1) antes de activar el temporizador?					
¿Cómo está A0.0 (bit T1) cuando se activa el temporizador?					
¿Cómo está A0.0 (bit T1) después de finalizar el tiempo de 8 s?					

¿Qué sucede en el tiempo del temporizador si antes de terminar los 8 s, desactivamos E0.0?					
¿Qué sucede si al concluir los 8 s, desactivamos y volvemos activar E0.0?					

TIPO	I	V	E	S	A
PROGRAMA	L 5T#8S U E0.0 SI T1 U T1 = A0.0	L 5T#8S U E0.0 SV T1 U T1 = A0.0	L 5T#8S U E0.0 SE T1 U T1 = A0.0	L 5T#8S U E0.0 SS T1 U T1 = A0.0	L 5T#8S U E0.0 SIA T1 U T1 = A0.0
¿Cómo está A0.0 (bit T1) antes de activar el temporizador?	0	0	0	0	0
¿Cómo está A0.0 (bit T1) cuando se activa el temporizador?	1	1	0	0	Al pulsar E0.0 la salida A0.0 pasa a 1, pero T1 arranca al soltar E0.0. Es decir que T1 se activa con la desactivación de E0.0
¿Cómo está A0.0 (bit T1) después de terminar el tiempo de 8 s?	0	0	1	1	0

¿Qué sucede en el tiempo del temporizador si antes de terminar los 8 s, desactivamos T1?	El tiempo se para y al volver a activar T1, se reinicia el tiempo	El tiempo no se para	El tiempo se para y al volver a activar T1, se reinicia el tiempo	El tiempo sigue, pero cuando llega a los 8 s, ya no se puede volver a arrancar. Para ello es necesario resetear T1. Añadir estas dos órdenes: U E0.1 R T1	El tiempo se para y al volver a activar T1, se reinicia el tiempo. Recordad que en este caso la activación de T1 se produce con la desconexión de E0.0
¿Qué sucede si al concluir los 8 s, desactivamos y volvemos activar E0.0?	Se reinicia el tiempo y arranca de nuevo	Se reinicia el tiempo y arranca de nuevo	Se reinicia el tiempo y arranca de nuevo	No vuelve a arrancar	Se reinicia el tiempo y arranca de nuevo

Se puede resumir diciendo que en los modos I y V la salida comienza activada y tarda el tiempo de temporización en desactivarse. En los modos E y S sucede lo contrario, la salida comienza desactivada y al concluir el tiempo, se activa. Asimismo, los modos I y E requieren tener activada la acción que los ha arrancado durante todo el tiempo de temporización porque, de lo contrario, se paran. En los modos V y S no pasa lo mismo, ya que no es necesario que la acción que los ha arrancado esté activada durante todo el tiempo.

Hay que destacar igualmente que **el modo S se debe resetear si se desea volver a arrancar.**

La instrucción de reset se puede utilizar para cualquier temporizador, ejecutándose la puesta a cero del tiempo de temporización.

Si se desea rearrancar un temporizador una vez que ha sido arrancado y la señal que lo ha activado sigue estando activa, existe la posibilidad de hacerlo sin necesidad de tocar el contacto o pulsador que lo arrancó. La instrucción que lo realiza es FR.

Un ejemplo: con E0.0 arranca el temporizador T1 y con E0.1 se rearranca sin necesidad de que se actúe sobre E0.0. En este caso será E0.1 el que realice la función de reiniciar el tiempo y arrancarlo.

```
U E0.0
L S5T#9s
SE T1

U T1
= A0.0

U E0.1
FR T1
```

En la Figura 74 hay un resumen de los diferentes modos de temporización. E0.0 es la acción que arranca el temporizador y A4.0 es la salida que activa y sigue al temporizador. El tiempo de temporización es *«t»*.

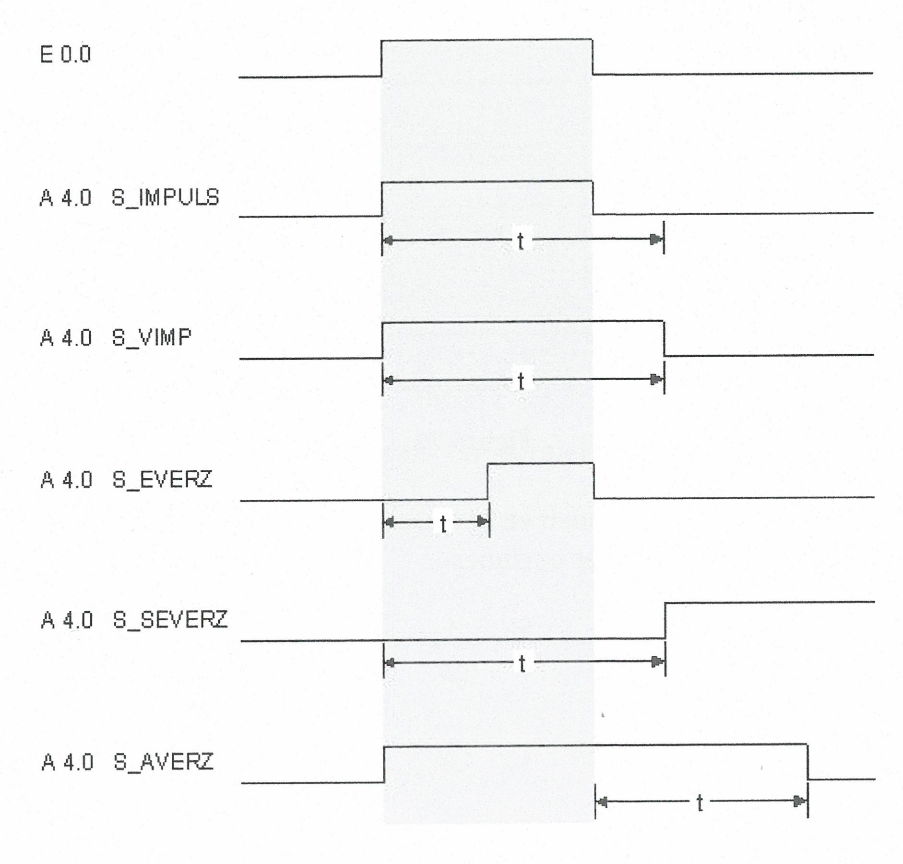


Figura 74

MARCA DE CICLO

Los autómatas de la serie S7 disponen de unas señales cíclicas. Son 8 señales de diferentes frecuencias que se llaman «marca de ciclo». Para poder utilizarlas hay que realizar el siguiente proceso:

1º. Ir al *hardware* y hacer clic dos veces (botón izquierdo del ratón) en el slot de la CPU. Se abrirá una ventana (**Propiedades**) de la que se selecciona la ficha **Ciclo/Marca de ciclo**.

2º. Se debe marcar la casilla **Marca de ciclo** e indicar un valor que utilizaremos como marca de ciclo. Podemos indicar cualquier cifra, pero no la podemos utilizar para nada más en el programa. Es una marca de byte.

3º. Se compila y se carga al PLC. Si no se compila, no surtirá efecto la marca de ciclo.

En la Figura 75 se indican los pasos que hemos descrito.

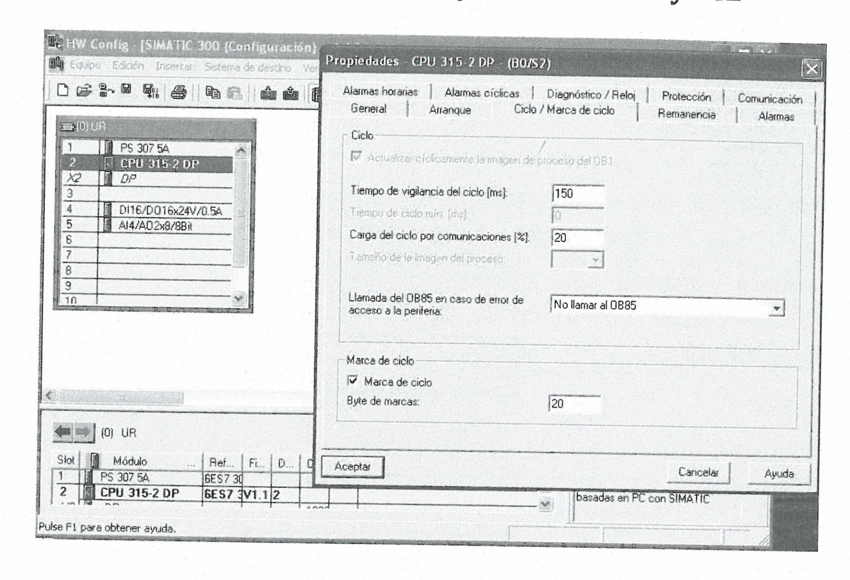


Figura 75

Las distintas frecuencias que se disponen en la marca de ciclo dependen del bit que se utilice. En la Figura 75.1 se incluyen diferentes opciones:

Bit	7	6	5	4	3	2	1	0
Duración del período (s):	2	1,6	1	0,8	0,5	0,4	0,2	0,1
Frecuencia (Hz):	0,5	0,625	1	1,25	2	2,5	5	10

Figura 75.1

Si se desea una salida intermitente de 1 segundo, se elige el bit 5. Como la marca que se ha seleccionado es la 20, el bit de marca para colocar en el programa y que sea de un tiempo de 1 s de intermitencia será M 20.5.

A continuación se realiza un ejemplo en el que una salida está intermitente con t = 1 segundo cuando un interruptor está cerrado. La marca de ciclo es 20.

```
U E0.0
U M20.5
= A0.0
```

Ejercicio resuelto

Se trata de comparar los valores de las dos entradas digitales, AB0 y AB1. Se realizará el programa que cumpla con la siguiente tabla:

EB0 < EB1	A0.0 activada fija
EB0 > EB1	A0.0 activada intermitente 1 s
EB0 = EB1	A0.0 activada intermitente 0,5 s

EB0 = 0000 1111	A0.1 activada
EB1= 1111 0000	A0.2 activada
EB0 = 0111 y EB1= 1110	A0.3 activada

L EB 0 L EB 1 <I = M 10.0 L EB 0 L EB 1 >I U M 20.5 = M 10.1 L EB 0 L EB 1 ==I	En los tres primeros casos la asignación es sobre A0.0. Para no repetir la misma salida, asignamos a unas marcas que luego ponemos en paralelo, para que cualquiera de ellas active A0.0.
U M 20.3 = M 10.2 U M 10.0 O M 10.1 O M 10.2 = A 0.0	Aquí se colocan las marcas en paralelo. La marca MB 20 es la marca de ciclo, M20.5 de 1 s y M20.3 de 0,5 s.
L EB 0 L 2#1111 ==I = A 0.1 L EB 1 L 2#11110000 ==I = A 0.2	

L EB 0 L 2#111 ==I = M 0.0 L EB 1 L 2#1110 ==I = M 0.1 U M 0.0 U M 0.1 = A 0.3	En este caso debe coincidir que EB0 sea 111 y EB1 1110, por lo que se asigna marcas a cada resultado y cuando las dos sean uno es que se cumple la condición establecida y se activa la salida A0.3.

COMPARACIONES CON TEMPORIZADORES

Aunque se dispone de suficientes temporizadores para poder utilizar uno por cada tiempo que se necesite, en alguna ocasión puede ser útil utilizar tiempos parciales de un mismo temporizador. Para ello se debería comparar con esos tiempos parciales. Para ello hay que tener en cuenta la forma en que se carga la palabra de temporización. El registro de temporización utiliza una palabra (W) en la que se incluye la base de tiempos y el valor del registro que va descontando para conseguir el tiempo deseado.

Para cargar un valor en un temporizador hemos visto que había dos formas. Una es mediante el valor estándar SIMATIC (L S5T#), en el que no se debe especificar la base de tiempos, ya que se elige automáticamente, y otra mediante la carga de la palabra (L W#16#) indicando la base de tiempos deseada y el tiempo en BCD. En función de cómo se cargue ese tiempo, la comparación se deberá hacer de forma diferente.

El siguiente ejemplo tratará este asunto. Dos salidas (A0.0 y A0.1) están conectadas inicialmente. Se carga un temporizador (T1) con un tiempo de 15 segundos y se desea que, cuando transcurran los 15 s, se desactive la salida A0.0 y cuando llegue a 5 segundos, se desactive la salida A0.1.

Como el temporizador decrementa su registro de temporización, habrá que restarle a 15 segundos los 5 s. En este caso, cuando el registro pase por 10 habrán pasado los 5 segundos deseados.

```
L W#16#2015
U E0.0
SV T1

U T1
= A0.0

LC T1
L 10
>=I
= A0.1
```

Si se carga el tiempo con la constante de tiempo SIMATIC, el método es similar. Como la base de tiempos la determina el programa, se tiene que conocer dicha base antes de utilizar la comparación. El valor máximo del registro de tiempo es 999. El sistema siempre intenta indicar el valor máximo con la base de tiempos más pequeña posible.

Por ejemplo, si el tiempo de temporización es de 15 segundos, el valor del registro será de 150 y la base de tiempos de 100 ms.......... 150 * 100 ms= 15 000 ms = 15 segundos.

Si fuera 1 hora, el registro sería de 1*60*60 s = 3600 segundos. El registro sería de 360 y la base de tiempos de 10 s............... 360*10 s= 3600 s = 1 hora.

Para comparar, se debería hacer lo mismo que antes, se tendría que restar al valor del registro para determinar el tiempo parcial que deseamos comparar.

Para el mismo ejercicio que antes:

```
L   S5T#15S
U   E 0.0
SV  T1

U   T1
=   A0.0

L   T1
L   100
>=I
=   A 0.1
```

Para terminar, conviene decir que el PLC 1500 dispone de una orden de comparar tiempos. Esto se verá en el apartado del PLC 1500.

EJERCICIOS CON TEMPORIZADORES

Los ejercicios 1, 2 y 6 se van a realizar de ejemplo.

1) Realizar un programa en el que se conecte un motor después de 4 segundos de cerrar un interruptor.
2) Hacer lo mismo, pero en este caso con un pulsador en lugar de un interruptor.
3) Realizar un programa que conecte tres motores en cascada, pero retardados 3 s a la conexión. Es decir, disponemos de un pulsador de marcha y activándolo arranca M1. Al cabo de 3 s arranca automáticamente M2 y al cabo de otros 3 s después de arrancar M2, lo hará M3. Disponemos de un pulsador de paro.
4) Realizar un lámpara intermitente de tiempo de cadencia 1 segundo.
5) Realizar el programa de mando de un arrancador estrella/triángulo para un motor asíncrono trifásico, t de estrella 2s.
6) Realizar el control de un semáforo en el que el de peatones se activa con un pulsador. Al activar el pulsador de peatones, el de coches se pone en ámbar y este tiempo dura 3 s. El tiempo de verde para peatones dura 6 s y el de rojo para coches 8 s. El semáforo de ámbar para peatones dura 2 s. Durante cada ciclo de peatones debe evitarse que se reinicie el tiempo al volver a accionar sobre el pulsador de peatones. Las condiciones iniciales son semáforo coches verde y semáforo peatones rojo.
7) Una fotocélula colocada en la puerta de una habitación detecta la entrada de personas y enciende automáticamente una lámpara. Esta lámpara solo se puede apagar con un pulsador. La habitación tiene tres ventanas. Si se entra por una de las tres, se enciende la luz y se dispara una alarma durante 10 s. Terminado el tiempo de la alarma, la luz se queda encendida y se puede apagar con un pulsador de reset. La alarma no se dispara si se abren las ventanas después de haber entrado por la puerta.
8) Una puerta se abre al hacer presión sobre una alfombra (solo en el lado de la apertura). Se queda abierta durante 8 segundos y luego se cierra automáticamente. Existe una fotocélula, de manera que cuando la puerta se está cerrando y pasa una persona (en cualquiera de los dos sentidos) se interrumpe la operación y la puerta se abre reiniciando el tiempo de espera. Habrá dos finales de carrera, uno para cuando la puerta esté cerrada y otro para cuando esté abierta.

EJERCICIOS RESUELTOS

1) Realizar un programa en el que se conecte un motor después de 4 segundos de cerrar un interruptor.

```
L   S5T#4S
U   E   0.0
SE  T   1

U   T   1
=   A   0.0
```

2) Hacer lo mismo, pero en este caso con un pulsador en lugar de un interruptor.

```
L   S5T#4S
U   E 0.0
SS  T1

U   T1
S   A0.0
R   T1          // el propio temporizador se resetea

U   E0.1        // para resetear la salida y volver a temporizar
R   A0.0
```

3) Realizar el control de un semáforo en el que el de peatones se activa con un pulsador. Al activar el pulsador de peatones, el de coches se pone en ámbar y este tiempo dura 3 s. El tiempo de verde para peatones dura 6 s y el de rojo para coches 8 s. El semáforo de ámbar para peatones dura 2 s. Durante cada ciclo de peatones debe evitarse que se reinicie el tiempo al volver a accionar sobre el pulsador de peatones. Las condiciones iniciales son semáforo coches verde y semáforo peatones rojo.

Se van a organizar las entradas y salidas; en este caso también pondremos símbolos a las entradas y salidas:

SALIDAS			ENTRADAS		
FUNCIÓN	DIRECCIÓN	SÍMBOLO	FUNCIÓN	DIRECCIÓN	SÍMBOLO
Verde peatones	A0.0	Vp	Pulsador	E0.0	Puls
Ámbar peatones	A0.1	Ap			
Rojo peatones	A0.2	Rp			
Verde coches	A1.0	Vc			
Ámbar coches	A1.1	Ac			
Rojo coches	A1.2	Rc			

Se ha realizado el programa utilizando los temporizadores en el modo V, impulso prolongado y utilizando solo asignaciones.

```
UN  "Rc"
UN  "Ac"     }  Si no está ni rojo ni ámbar para coches, se pone verde.
=   "Vc"

UN  "Vp"
UN  "Ap"     }  Si no está ni rojo ni ámbar para peatones, se pone verde.
=   "Rp"
```

```
U   "P"
UN  "Ac"
UN  "Rc"
UN  "Vp"
L   S5T#3S
SV  T1

U   T1
=   "Ac"
```

Al pulsar, y, si no está rojo ni ámbar para coches y ni verde para peatones, se pone ámbar para coches durante 3 s. UN Ac impide que, al volver a pulsar, se reinicie el tiempo.

```
U   "Ac"
FN  M0.0
=   M0.1
```

Con el flanco negativo conseguimos arrancar T2 y T3 cuando termina el ámbar coches.

```
U   M  0.1
L   S5T#8S
SV  T2

U   T2
=   "Rc"
```

Cuando termina ámbar para coches, se pone rojo para coches 8 s.

```
U   M  0.1
L   S5T#6S
SV  T  3

U   T  3
=   "Vp"
```

Cuando termina ámbar para coches, se pone verde para peatones 6 s.

```
U   "Vp"
FN  M  0.2 .
L   S5T#2S
SV  T  4

U   T  4
=   "Ap"
```

Con el flanco negativo conseguimos arrancar el T4 cuando termine el verde peatones. Cuando termina verde para peatones, se pone ámbar peatones 2 s.

CONTADORES

Los contadores son elementos que cuentan eventos externos, es decir, impulsos en una entrada del autómata. Se puede decir que son como los temporizadores, ya que estos cuentan impulsos de una base de tiempos interna.

Los contadores consisten en un registro de palabras (16 bits) que puede incrementarse o decrementarse. Se identifican con la letra Z (Z0.....Z63). También se pueden cargar en un valor determinado y, a partir de él, decrementar o incrementar, según nos interese. Pueden igualmente borrarse (reset) y ser consultados, a nivel de bit o del valor del registro de contaje. Los contadores, si no se carga un valor previamente, parten inicialmente de cero. El valor máximo que se puede contar con cada uno de los contadores es 999 y el mínimo 0. No se da la vuelta al llegar al máximo ni al mínimo, se quedan en 999 o 0 respectivamente.

La instrucción que hace que el contador cuente en **sentido creciente** es **ZV**, poniendo como operando el contador que se desea utilizar, por ejemplo, ZV Z5. Para que se incremente Z5, el RLO debe pasar de 0 a 1 (flanco de subida).

Las instrucciones para comenzar a utilizar un contador que comience en su valor inicial por defecto, es decir que empiece a contar desde cero, son estas (para el contador Z5, por ejemplo):

```
U E0.0
ZV  Z5
```

Con cada activación de E0.0, el registro de Z5 se incrementa en una unidad.

La instrucción que hace que el contador cuente en **sentido decreciente** es **ZR**, poniendo como operando el contador que se desea utilizar, por ejemplo, ZR Z5. Para que se decremente Z5, el RLO debe pasar de 0 a 1 (flanco de subida).

Para descontar las instrucciones:

```
U E0.1
ZR  Z5
```

Si lo que se desea es que el contador comience desde un **valor previo,** la instrucción que se debe indicar es **L C#valor**, donde **valor** es el número con el que se desea que comience el contador. Con esa instrucción cargamos en el acumulador 1 (ACU 1) el valor, después se debe cargar el valor de ACU1 en el contador correspondiente y la orden para ello es SET, **S Z5.** Por ejemplo:

```
U E0.2
L C #20
S Z5
```

Con esas instrucciones, al accionar E0.2 carga el valor 20 en el contador Z5. Se debe recordar que la orden de carga (**L**) es incondicional, por lo que es lo mismo incluirla antes que después de U E0.2.

Para **borrar** un contador, simplemente hay que utilizar la orden de reset indicando el contador que se desea borrar. Por ejemplo:

```
U E0.3
R Z5
```

Para poder saber cómo está el contador se disponen de varios métodos. Uno de ellos sería **consultar el** estado **del bit** del contador. El bit se indica con el identificador del contador (Z5 por ejemplo) y utilizando una instrucción de bit. Este bit será 0 cuando el contador llegue o

esté a cero. Cualquier otro valor distinto de cero dará como resultado que el bit del contador esté a 1. De este modo se puede saber cuándo el contador llega a cero. Si se desea que se active una salida (A0.0) cuando el contador llegue a cero, se escribiría así:

```
UN Z5
= A0.0
```

Otra forma de poder **consultar** el estado del contador es con **comparaciones**: se compara el contador con un valor concreto. Por ejemplo, se desea que se active la salida A0.0 cuando el contador llegue a 10:

```
L Z1
L 10
== I
= A0.0
```

EJERCICIO RESUELTO

1. Se trata de realizar un programa que incluya tres opciones de carga de un mismo contador. Una vez seleccionado el valor de carga, ya no se podrá cargar otro valor hasta que no se resetee el contador. El contador se incrementa con la detección del paso de botellas mediante un detector óptico. No debe empezar a contar hasta que no se haya preseleccionado el valor inicial. Los valores a precargar son 10, 20 y 40. Se debe indicar mediante una lámpara cuando llegue a cero y con otra lámpara cuando el valor pase de 9.

```
U    E0.0
UN   M0.0
L    C#10
S    Z1
S    M0.0

U    E0.1
UN   M0.0
L    C#20

S    Z1
S    M0.0

U    E0.2
UN   M0.0
L    C#40
S    Z1
S    M0.0
```

Con E0.0, E0.1 y E0.2 se seleccionan los valores de carga. El primer valor cargado pondrá la marca M0.0 a uno, y hará que se abra e impida que al volver a pulsar cualquiera de esos pulsadores se active la carga de nuevo. Además, esa marca va a servir para impedir que se pueda decrementar el contador en el bloque siguiente.

```
U    M0.0
U    E0.3
ZR   Z1
```

Hasta que no se cierre M0.0, no podremos decrementar el contador y esto solo ocurrirá activando la carga anterior.

```
UN   Z1
=    A0.0
```

A0.0 indica que el contador es cero.

```
L   Z1
L   9                A0.1 indica que el contador es mayor de 9.
>I
=   A0.1
U   E0.4
R   Z1               E 0.4 resetea el contador y marca M0.0.
R   M0.0
```

EJERCICIOS PARA RESOLVER

1. Se desea realizar un programa que automatice un garaje de diez plazas. Cuando esté lleno, se encenderá un cartel indicando COMPLETO y la barrera ya no se levantará hasta que salga algún coche y haya alguna plaza libre. Cuando haya alguna plaza libre, se encenderá un cartel que indicará LIBRE.

Se dispone de dos células fotoeléctricas, una en la entrada y otra en la salida.

2. Se trata de un almacén intermedio. Dispone de dos cintas transportadoras, una de entrada de producto y otra de salida. En la entrada del almacén (final de la cinta de entrada de producto) hay una fotocélula que cuenta los productos que entran. La segunda cinta lleva los productos a los diferentes puntos de distribución. También en la salida del almacén (principio de la segunda cinta) existe otra fotocélula para conocer los productos que salen del almacén intermedio.

Hay unas lámparas que indican la carga del almacén, según la tabla siguiente:

Almacén	VACÍO	NO VACÍO	AL 50 %	Más del 90 %	LLENO
Salida	A0.0	A0.1	A0.2	A0.3	A0.4

El programa debe realizar la conexión de las lámparas según la tabla anterior. Igualmente, parará la cinta de entrada de producto si ya está lleno y parará la cinta de salida de producto si está vacío.

3. Se trata de programar una escalera automática que se conecta cuando detecta el paso de personas. Se dispone de un interruptor general que conecta la instalación. Cuando una fotocélula detecta el paso de una persona, la escalera se pone en marcha. A los 15 segundos se para. Si se detecta alguna otra persona antes de los 15 segundos, el tiempo se reinicia.

Hay dos pulsadores/seta de emergencia, uno al principio y otro al final de la escalera.

La escalera se puede parar por varias causas:

- Al pulsar alguna seta de emergencia, para ponerla en marcha se debe levantar la seta y activar el pulsador de rearme.
- Si se dispara el relé térmico, se vuelve a poner en marcha automáticamente cuando el relé se restablece por sí solo.
- Al desconectar el interruptor general, se pone de nuevo en marcha al volver a conectarlo y detectar una persona.

- Cuando el relé térmico se dispara tres veces, la escalera se bloquea y ya no la pone en marcha el simple restablecimiento del relé térmico. Para poner en marcha la instalación, existe un pulsador especial de desbloqueo.

4. Se trata de realizar un programa que cuente los impulsos positivos de una señal fija de periodo seleccionable mediante pulsadores (se debe utilizar marca de ciclo) y los impulsos de una entrada a la que se le conecta un detector. Los tiempos serán de 0,5 s, 1 s y 2 s. Estas cuatro opciones (los tres tiempos y la entrada del detector) se podrán seleccionar mediante interruptores. Con un pulsador se puede hacer que se inicie el conteo creciente, con otro el conteo decreciente y con un tercer pulsador se podrá resetear el contador. Se pueden cargar tres valores diferentes seleccionados, cada uno, con un pulsador diferente. Los valores de carga son 10, 30 y 50.

5. Hay que automatizar el arranque de tres motores. Estos motores se mantienen en marcha durante un tiempo determinado. Disponen de tres ciclos diferentes de tiempos. Cada ciclo se puede activar mediante el estado de un contador que es incrementado o decrementado mediante sendos pulsadores y se pasará de un ciclo a otro cíclicamente mediante la activación de un pulsador que incremente el contador u otro pulsador que lo decremente. El ciclo no podrá cambiarse mientras los motores no se paren. Un pulsador parará los motores.

Los ciclos y sus tiempos son:

CICLO	MOTOR 1	MOTOR 2	MOTOR 3
1	5 s	8 s	20 s
2	7 s	6 s	8 s
3	4 s	12 s	15 s

Se debe emplear lista de saltos (SPL) y los tiempos de cada ciclo se pueden hacer con un mismo temporizador comparando o con tres temporizadores distintos.

6. Se desea realizar un marcador de baloncesto mediante 5 pulsadores, de forma que se realicen las siguientes operaciones:

Con una entrada se añade un punto al valor almacenado. Con otra entrada se añaden dos puntos al marcador y con una tercera entrada tres puntos.

Otro pulsador modifica el valor del marcador decrementándolo un punto. Aún hay otro pulsador que pone a cero el marcador.

7. Se dispone de dos entradas: con una se incrementa un registro y con la otra se decrementa. Cuando llega a 10, se conecta una salida y se impide que siga contando. El registro podrá seguir contando cuando se decremente en una unidad como mínimo, momento que debe apagarse la salida.

8. DIAGNOSIS

INTRODUCCIÓN

Cuando existe un problema en la CPU o cuando un programa terminado no funciona correctamente, es necesario disponer de alguna herramienta que facilite la resolución del problema. En los temas anteriores ya se ha visto alguna forma de ayudar a revisar lo realizado. La tabla de variables es una de esas ayudas con las que se puede revisar el comportamiento de las variables en tiempo real. Lo mismo sucede con la opción de las «gafas» en AWL y KOP, con la que se puede realizar un seguimiento del estado del programa.

Todo esto es diferente si se trata de STEP 7 V5.5 o TIA PORTAL y por eso se estudian las dos plataformas.

DIAGNÓSTICO STEP7 V5.5

BUFFER DE DIAGNÓSTICO

Los métodos enumerados anteriormente son soluciones a nivel de *software*. El programa Step 7 dispone soluciones de ayuda para problemas de *hardware*. Para ello se hará uso del buffer de diagnóstico que facilita información sobre lo que sucede, tanto a nivel de *software* como de *hardware*. Es una zona de memoria donde se guarda información de lo que se realiza con el autómata: arrancar, poner en stop, si faltan módulos por cargar, si fallan direcciones, etc. Se accede a él desde varios lugares, por ejemplo, desde la ventana **Hardware**, tocando y resaltando la CPU, tal como aparece en la Figura 76.

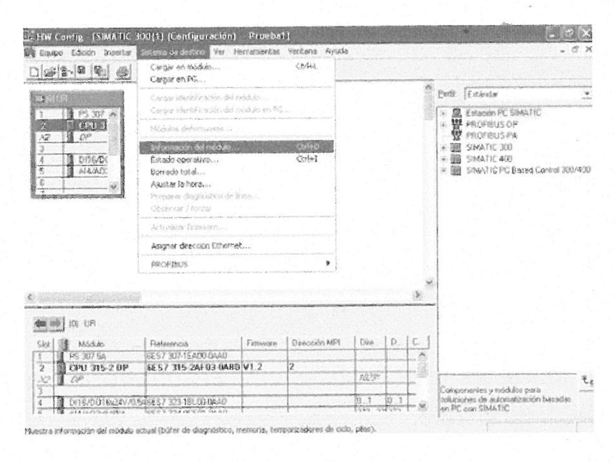


Figura 76

En la pestaña **Búfer de diagnóstico** se puede leer lo que ha sucedido, tal como se muestra en la Figura 77.

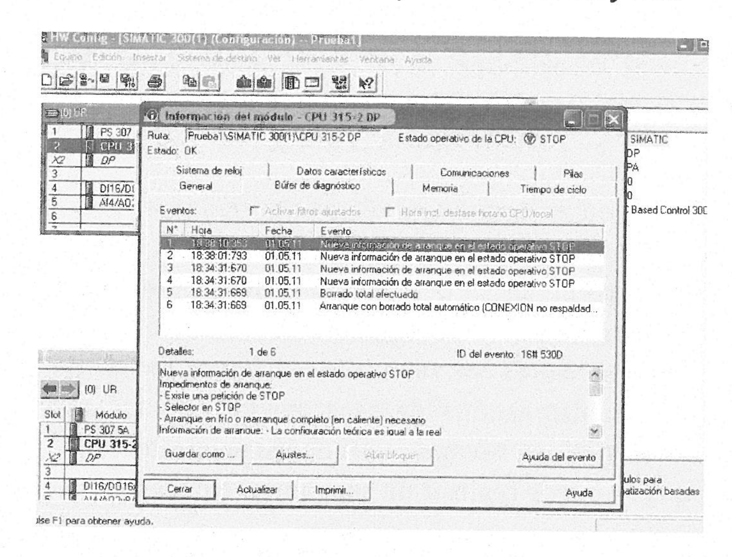


Figura 77

También puede facilitar información de diagnóstico respecto a otros módulos como las E/S. Si se toca con el ratón sobre el módulo de E/S digitales, se observa que igualmente ofrece información de diagnóstico, y se puede comprobar si hay algún error e intentar solucionarlo. La Figura 78 indica lo que nos dice de un módulo concreto:

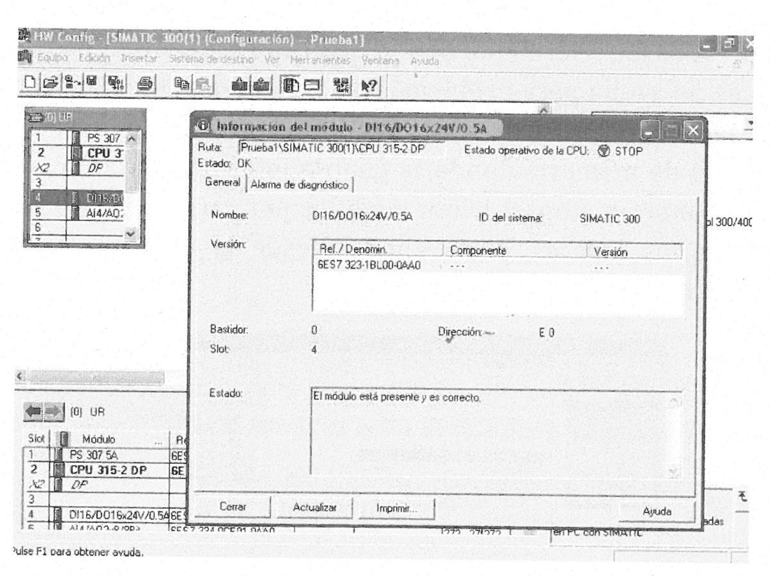


Figura 78

Vamos a ver un ejemplo donde se puede observar claramente lo importante y cómodo que resulta el uso del buffer de diagnóstico.

Hay que copiar este programa en OB1: (FC1 es una función, pero no es necesario saber ahora qué es una función).

```
U E0.0
CC FC1

U E0.1
= A0.0
```

Hay que arrancar el PLC (o el simulador) y activar la entrada E0.0. Inmediatamente se puede ver que el led SF se pone en rojo y el PLC se va a STOP. Para comprobar qué es lo que está pasando se debe abrir **Búfer de diagnóstico** e intentar descubrir lo que sucede.

En la Figura 79 se detalla la información que facilita.

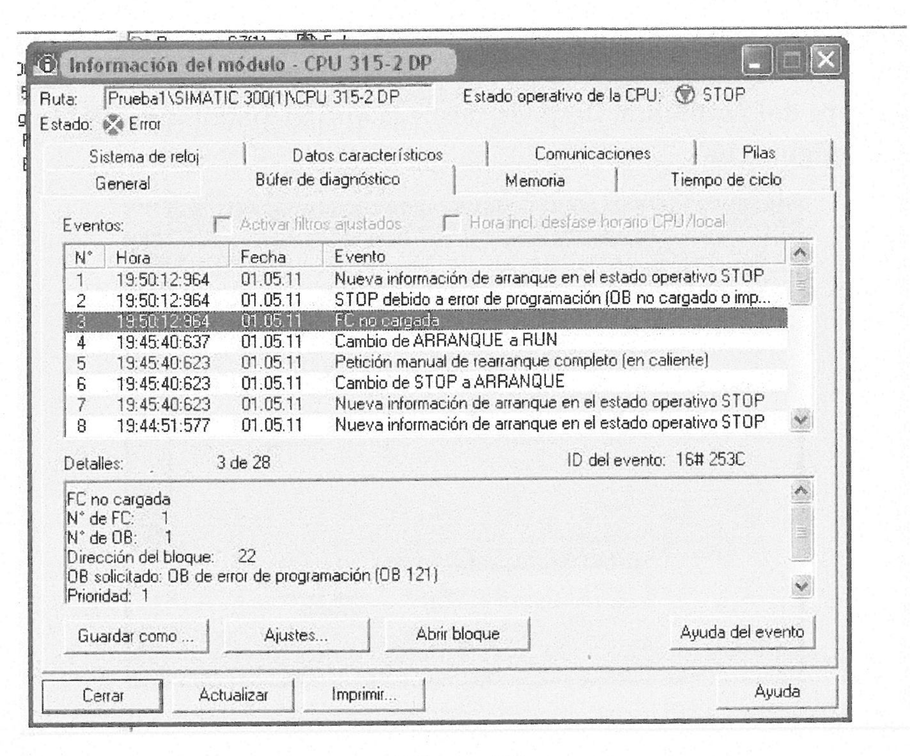


Figura 79

Se puede ver la solución que ofrece: **No se ha cargado FC1 en el PLC.** El problema se soluciona añadiendo el módulo de *software* FC1 y cargándolo también al PLC. Al volver a arrancar, se soluciona el problema. Para incluir la FC1 se debe ir a los bloques e insertar un nuevo objeto que se llama FUNCIÓN. La Figura 80 indica cómo colocar el módulo de función.

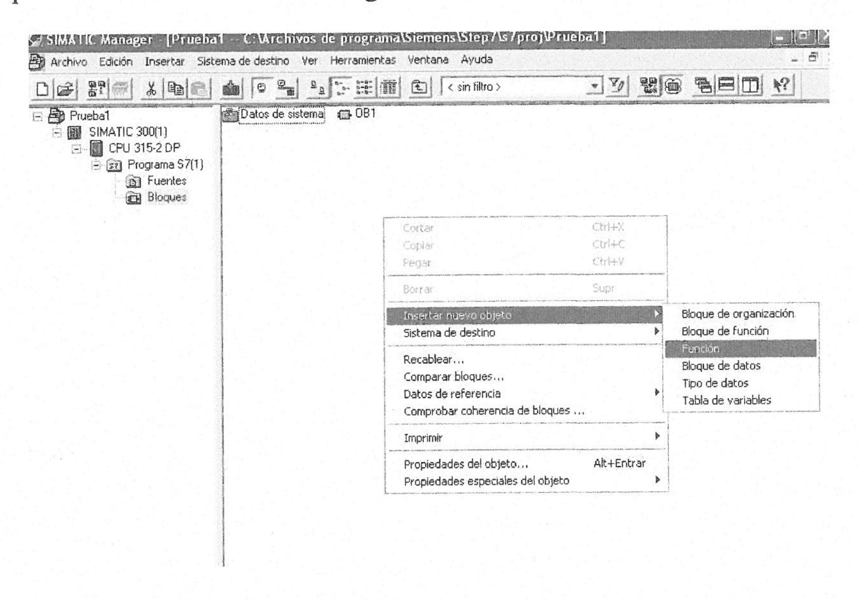


Figura 80

Una vez colocado en bloque, se envía. No es necesario escribir nada en la función, solo enviarla al PLC.

Ahora, al activar E0.0, se comprueba que todo es correcto, es decir, que no hay ningún led de error encendido.

Como se ha mencionado, hay otras formas de acceder al diagnóstico. Desde el administrador, podemos observar el menú que aparece en la Figura 81. Para acceder a esta pantalla se debe tocar en el **nombre del autómata**. Después, desde la opción **Sistema de destino** se accede a **Diagnóstico/Configuración**.

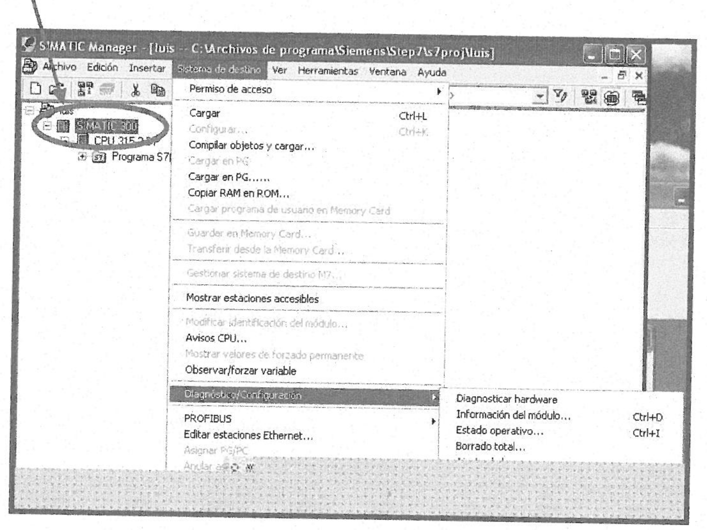


Figura 81

Si se activa **Diagnosticar hardware,** se puede entrar también en el buffer de diagnóstico seleccionando **Información del módulo**. Si se elige **Abrir equipo online,** se accede a la ventana de *hardware* de STEP 7, pero en modo online, para ver el estado del PLC en tiempo real. Desde aquí también se puede comprobar el estado del PLC.

En la Figura 82 se muestra el contenido de esta ventana.

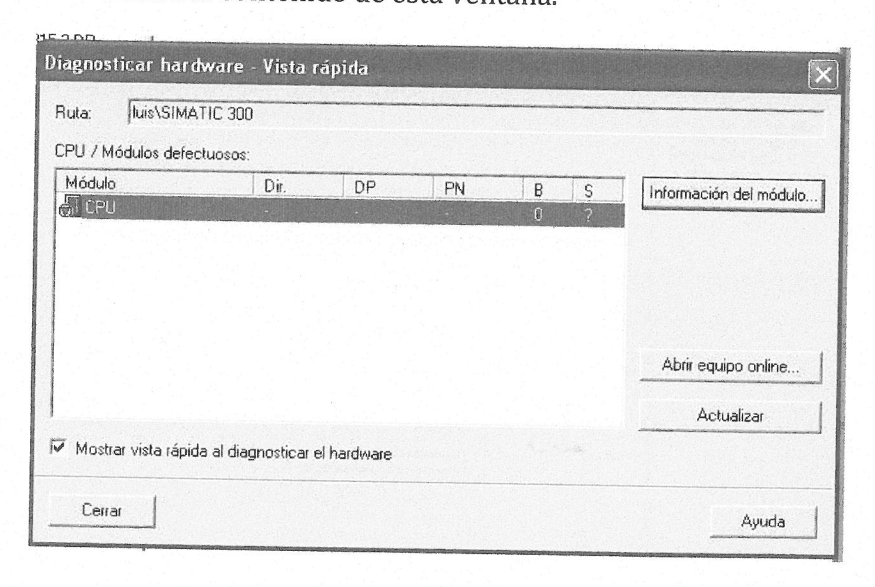


Figura 82

También se puede cambiar el PLC en estado online/offline activando el icono desde la ventana *hardware*.

Siguiendo en el menú **Diagnóstico/Configuración,** si se selecciona **Información del módulo,** se accede al buffer de diagnóstico. Desde el estado operativo se puede detener/arrancar el PLC (Figura 83).

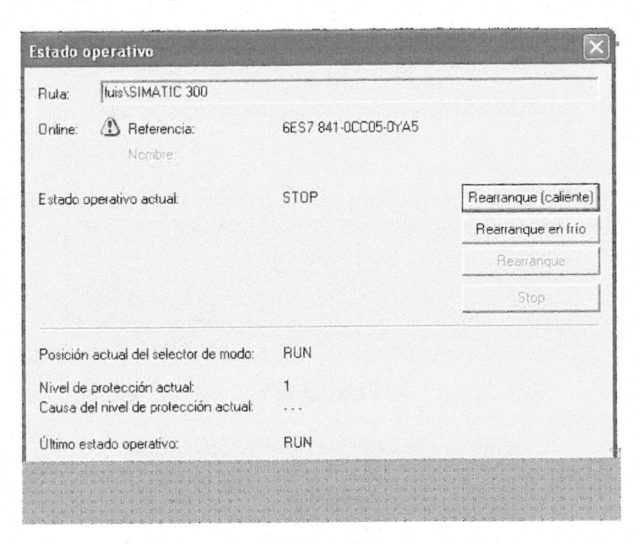


Figura 83

La opción **Borrado total** elimina la mayoría de la información que tiene el PLC, incluidos los datos de la configuración del *hardware* y el propio programa. Esta operación es útil llevarla a cabo cuando el PLC se comporta de forma extraña.

CARGAR EN PROGRAMADORA (PC/PG)

La opción cargar en PG realiza la carga de los datos de configuración y del programa del PLC sobre el PC. Dejará en Step 7 el contenido que tenga el PLC, lo que puede ser útil para tareas de mantenimiento o edición de programas, sobre todo si no han sido realizados por el personal de mantenimiento.

Se accede desde el menú principal (**Administrador**) si es que se quiere cargar todo. También se puede acceder desde **Hardware,** pero en este caso solo se cargarán los datos de la configuración.

Para cargar, se debe abrir un proyecto nuevo y asignarle un nombre. Después, desde la opción **Sistema de destino** se selecciona **Cargar en PG**. En el proceso de carga se solicitará la dirección MPI para poder acceder al PLC. Si se desconoce, debemos hacer clic en **Mostrar** para que aparezca dicha dirección. Hemos de **Aceptar** y hacer lo mismo con todos los mensajes que se muestren (Figura 84).

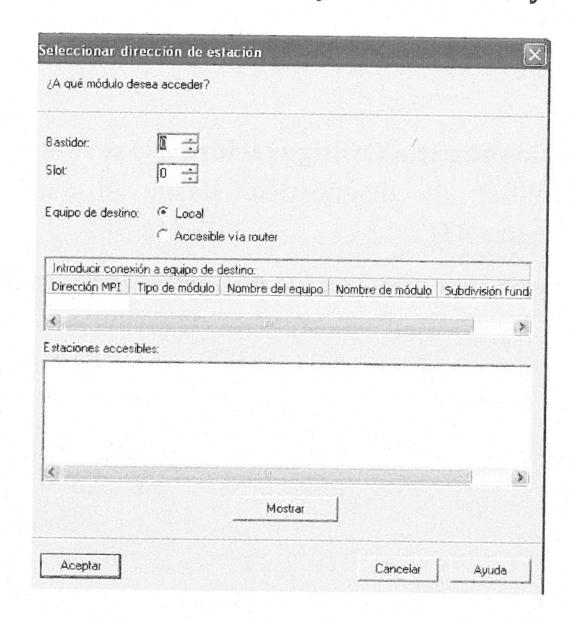


Figura 84

REFERENCIAS CRUZADAS

Desde el punto de vista de la edición de un programa, es importante poder localizar de una forma rápida el lugar en el que se encuentran las variables. En el editor de programas se dispone de la opción **Referencias cruzadas**, que se encuentra en la parte inferior. Ofrece información sobre las variables que se utilizan, indicando el lugar en el que se encuentran y su función: si es asignación, si está «seteada», si es entrada, etc. Esto viene muy bien para observar si se han repetido asignaciones de una misma salida, por ejemplo. Para poder ver las referencias cruzadas es necesario haber grabado antes el programa. En la Figura 85 se muestra un ejemplo.

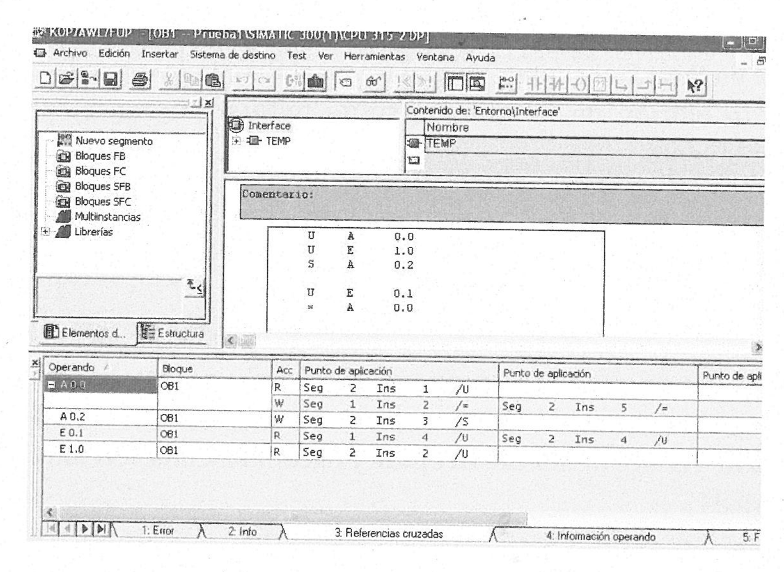


Figura 85

DIAGNOSTICO TIA PORTAL V13

BUFFER DE DIAGNÓSTICO

En TIA PORTAL el acceso al buffer de diagnóstico se realiza desde la posición online. Son muchas las ocasiones en las que es necesario conectarse online con el sistema de control, por ejemplo:

- Comprobar el estado real del sistema, hacer un diagnóstico y determinar qué es lo que puede fallar.
- En modo online se puede comprobar el estado real de la configuración de cada componente del sistema.
- También se pueden modificar y cargar en cada dispositivo determinados valores desde la opción **Funciones**.

En las siguientes figuras se muestra cómo proceder.

Desde el árbol del proyecto y en **Accesos Online**, se selecciona la interfaz que se está utilizando, en este caso la tarjeta de red REALTEK. Ver Figura 86.

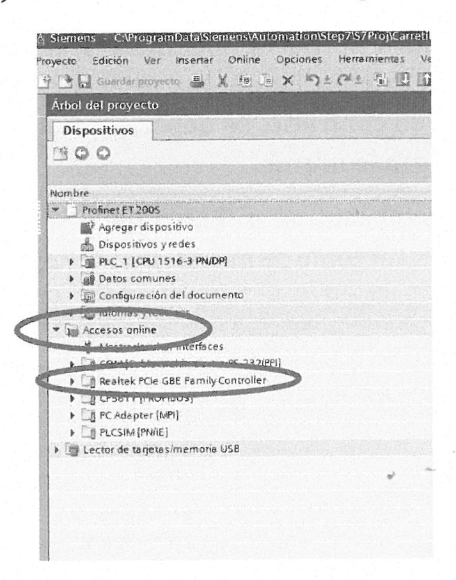


Figura 86

Se pulsa en **Actualizar dispositivos accesibles** y esperamos:

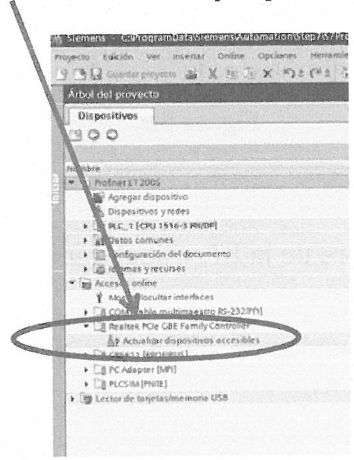


Figura 87

Al poco tiempo se mostrarán todos los dispositivos accesibles:

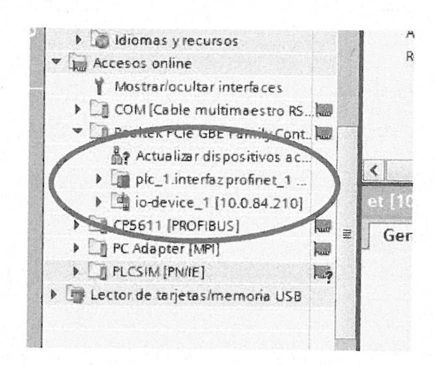


Figura 88

Se marca el dispositivo que queremos online y seleccionamos **Online y diagnóstico**. Desde aquí se podrá hacer el diagnóstico, además de cambiar la dirección IP y el nombre del dispositivo.

Se puede realizar el diagnóstico, y se puede modificar y enviar una nueva IP y un nuevo nombre. Pero, ATENCIÓN: si se cambia, habrá que estar seguro de disponer de los mismos datos en el proyecto offline. De lo contrario, se deberán cambiar para que, tanto el proyecto offline como el real (*online*), tengan lo mismo.

Las opciones de modificación se encuentran en **Funciones**. Allí se podrán cambiar y enviar las IP u otras direcciones y nombres, tal como se muestra en la Figura 89.

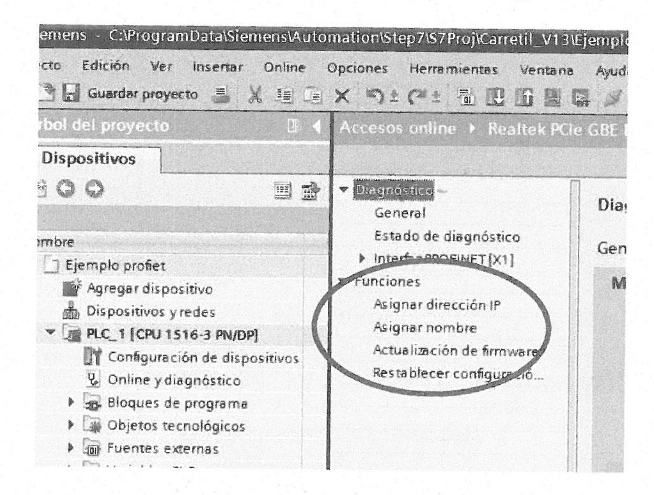


Figura 89

Aquí también se podrá actualizar el firmware una vez bajado el fichero correspondiente.

Por ejemplo, para cambiar el nombre de la ET, trabajaremos en la ventana que se representa en la Figura 90.

Estas opciones pueden variar dependiendo del dispositivo del que se trate, además de la versión y actualización de TIA PORTAL.

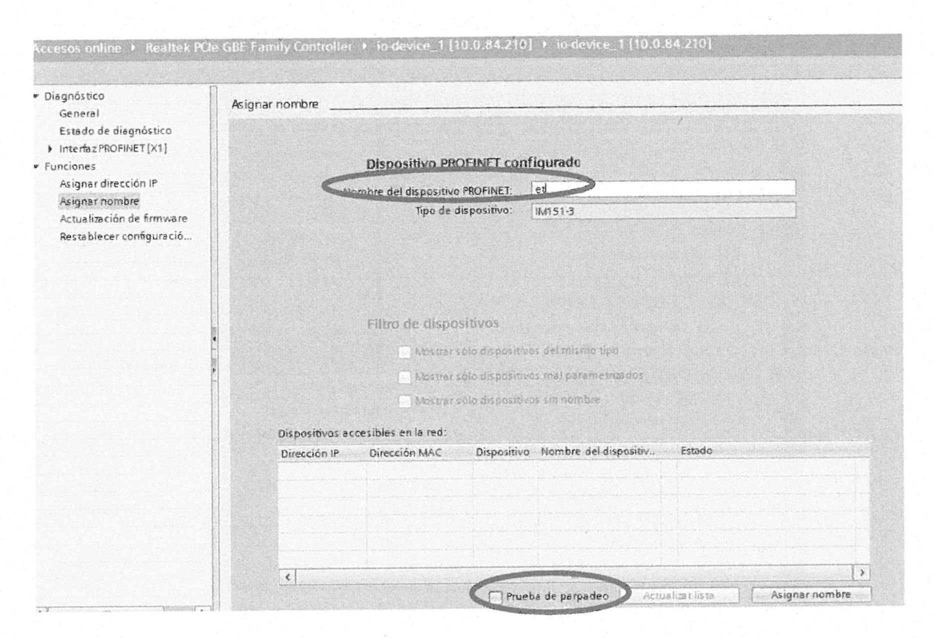


Figura 90

Para salir del modo Online basta con cerrar la ventana.

REFERENCIAS CRUZADAS

También en TIA PORTAL se pueden encontrar las referencias de todas las variables que se hayan colocado en el programa. Para acceder a ellas basta con situarse con el ratón en **Bloques de programa** y seleccionar **Referencias cruzadas** con el botón derecho (Figura 91).

Una vez se ha abierto en la pantalla la ventana de las referencias cruzadas, aparecen todos los bloques de programa que se utilizan en el proyecto. Se selecciona la pestaña **Utilizado** y se abre el bloque en el que se encuentra la variable de la que se quieren determinar las referencias cruzadas. También se muestran los lugares en los que se ha utilizado y el tipo de acceso: lectura o escritura (Figura 92).

Otra forma de acceder a las referencias cruzadas es desde el mismo bloque de programación. Se sitúa el ratón sobre la variable a referenciar y, en la parte inferior, en la pestaña de **Referencias cruzadas** se observará los lugares donde se encuentra utilizada dicha variable.

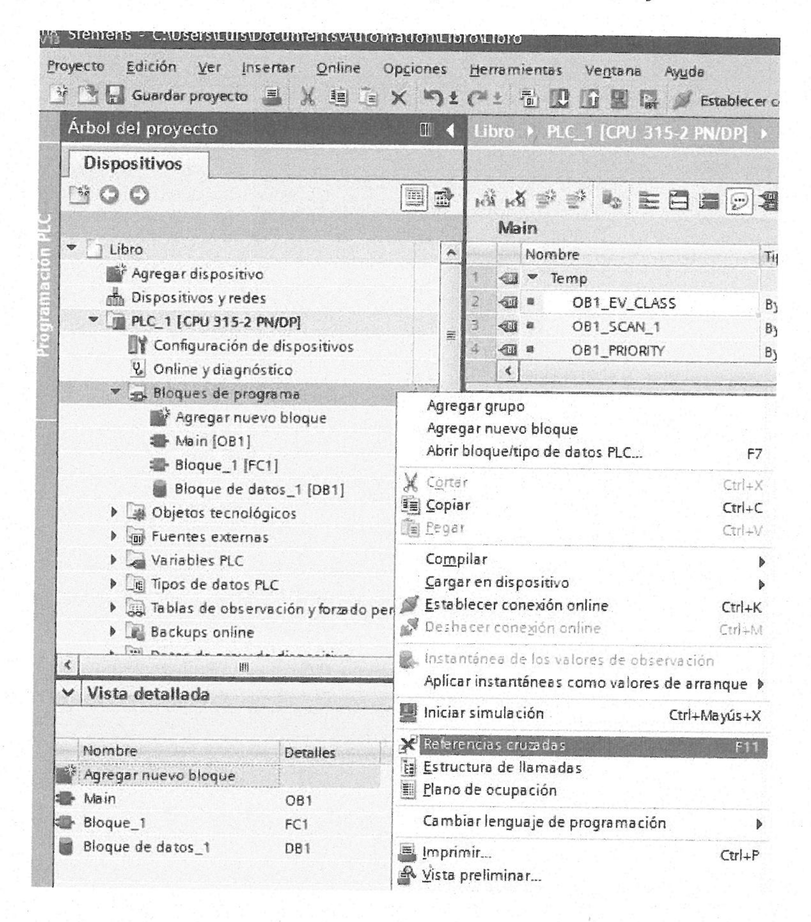


Figura 91

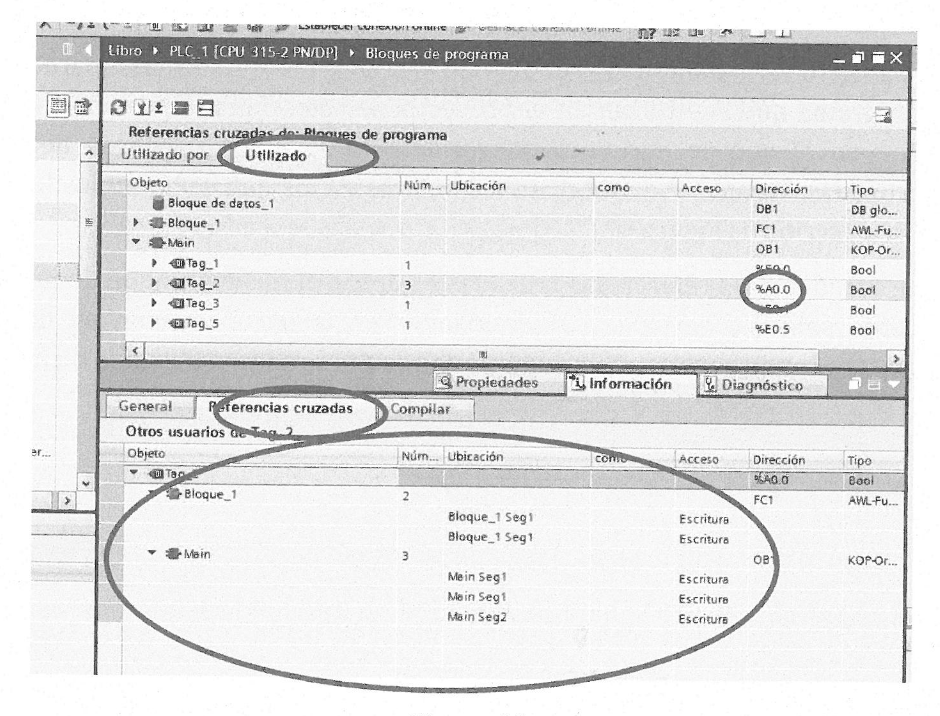


Figura 92

9. ENTRADAS/SALIDAS ANALÓGICAS

INTRODUCCIÓN

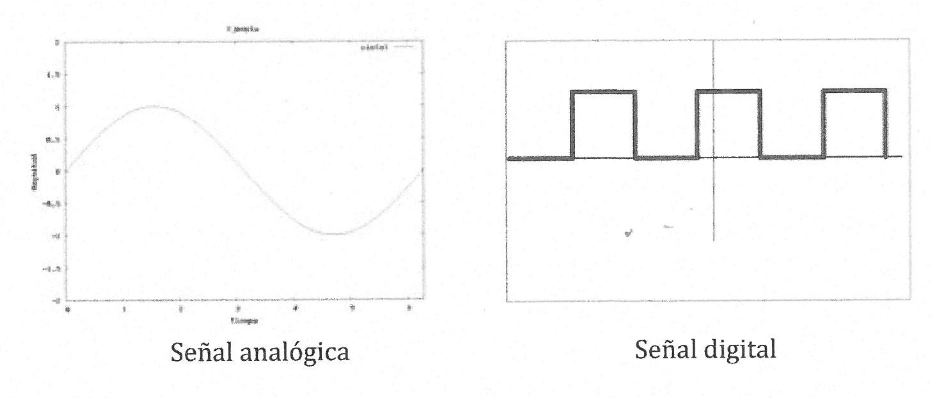
Es sabido que todos los sistemas que procesan información son digitales, es decir, que solo «entienden» unos y ceros. Puede parecer anticuado y anacrónico que en la era digital se hable de que algo sea analógico, pero la realidad está empeñada en demostrar todo lo contrario, y hablar de «lo analógico» en la industria es de vital importancia.

En este tema se van a aclarar estos conceptos, así como el tratamiento de las señales analógicas por el PLC.

CONCEPTOS

Lo primero que se debe conocer es la diferencia entre una señal analógica y una digital. Una señal analógica es aquella que puede representarse mediante una función continua, y se caracteriza por su periodo y amplitud.

Los valores digitales son aquellos que pueden representarse mediante valores discretos. En la Figura 93 se distinguen una señal analógica y una digital.



Señal analógica Señal digital

Figura 93

Son muchas las ventajas de los sistemas digitales frente a los analógicas. Por ello, es evidente que, a la vista de cómo todo se está «pasando» a lo digital, los sistemas basados en tecnologías digitales ganen terreno a lo analógico. Algunas de estas ventajas son:

- ✓ Capacidad de almacenamiento de grandes cantidades de información.
- ✓ Procesamiento de la información.
- ✓ Facilidad, o por lo menos mayor efectividad, de transporte, ya que no se envían señales analógicas, que son fáciles de perturbar. Se envían paquetes de información.
- ✓ Comprobación sencilla de que la información recibida es la que se ha enviado.

Son muchos los ejemplos que se pueden incluir de aplicaciones digitales. La TV digital es un claro ejemplo de las ventajas del uso de los sistemas digitales, además de las cámaras digitales, el ordenador, etc.

Uno de los principales inconvenientes de los sistemas digitales es la incapacidad de manejar información de tipo analógico de forma directa. Si se piensa en que todas las magnitudes físicas son analógicas, se observa la importancia de poder procesar información analógica. El no poder hacerlo sería tanto como decir que no se puede procesar la temperatura, la aceleración, la velocidad, la presión, etc. y, por lo tanto, no se podrían realizar controles de estas magnitudes mediante un ordenador o un autómata.

Este problema se soluciona digitalizando las señales analógicas, un proceso que consiste en tomar la señal y trocearla, coger paquetes de señal. A cada paquete de esa señal se le asigna un valor digital y así podemos introducir la señal a un sistema digital y procesarla. En la Figura 94 se aprecia cómo la señal analógica se «trocea en paquetes» de señal.

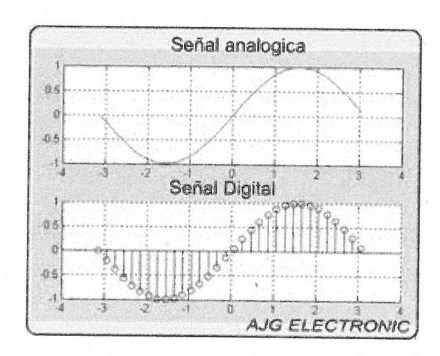


Figura 94

Es evidente que, cuantos más trozos se hagan de la señal, mejor, más fiel a la señal analógica será la digitalización. El problema es económico, ya que cuantos más trozos podamos crear en la señal, más caro resulta el proceso de digitalización. Este proceso lo realizan los circuitos integrados llamados conversores de digital a analógico y viceversa (DAC y ADC). La señal hay que prepararla antes de digitalizarla. El proceso completo de digitación lo realizan las tarjetas de entradas y salidas analógicas. Una de las características de estas tarjetas está asociada al número de paquetes que pueden generar en la señal analógica y esto está íntimamente relacionado con el número de bits del que dispone la tarjeta para digitalizar. Como se ha comentado, la tarjeta debe trocear la señal y a cada trozo le asigna un valor binario. Según este proceso, primero lee un valor analógico y después le asigna un valor digital. El valor digital se guarda en un registro interno que se encuentra en la memoria de la tarjeta. Con una tarjeta de 2 bits tan solo se pueden realizar 2^2 paquetes de valores digitales, es decir, 4. En la señal senoidal de la Figura 94, si muestreamos solo 4 puntos, la fidelidad de la señal digitalizada estará muy alejada de la realidad. Con 4 bits se pueden obtener $2^4 = 32$ puntos de muestras, con 8 bits $2^8 = 256$ y cuantos más bits, más puntos y, por lo tanto, más resolución. Y también un mayor precio, ya que los circuitos integrados, y por tanto las tarjetas que los llevan, se encarecen con el número de bits. Se tendrá que ponderar la relación resolución/precio en función de la aplicación donde se vaya a utilizar.

TARJETAS DE ENTRADA Y SALIDAS ANALÓGICAS

CARACTERÍSTICAS

Las tarjetas que se utilizan en los PLC se caracterizan por el número de entradas y salidas analógicas que disponen y por el número de bits que emplean en la digitalización.

Para el PLC de la serie S7 300, se va a utilizar para la explicación la tarjeta SM 334 AI4/AO2 de 8 bits (Ref. 334 0CE01 0AA0). Tal como indican sus siglas, es una tarjeta de 4 entradas analógicas (AI 4) y 2 salidas analógicas (AO2). La resolución de la tarjeta es de 8 bits. Puede digitalizar tensiones o corriente. Los rangos son de 0 a 10 V para la tensión y 0 a 20 mA para las corrientes. En la Figura 95 se puede apreciar una tarjeta de estas características.

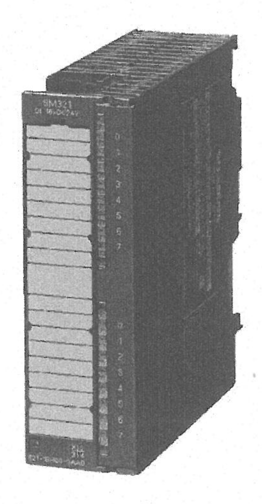


Figura 95

En la siguiente tabla se detallan sus características más significativas:

Cantidad de entradas	4 entradas formando 1 grupo de canales 4
Cantidad de salidas	2 salidas formando 1 grupo de canales
Resolución	8 bits
Tipo de medición	Ajustable por cada grupo de canales: • Tensión • Intensidad
Tipo de salida	En cada canal: • Tensión • Intensidad
Soporta operación sincronizada	No
Diagnóstico parametrizable	No
Alarma de diagnóstico	No
Supervisión de valores límite	No
Alarma de proceso rebase valor límite	No
Alarma de proceso fin de ciclo	No
Emisión valores sustitutivos	No
Relaciones de potencial	Enlace galvánico con la CPU • Separación galvánica con la tensión de carga
Particularidades	No parametrizable; ajuste del tipo de medición y de salida mediante cableado

CONEXIONADO

En la Figura 96 se muestra el conexionado de la tarjeta. Los terminales 2, 5, 8 y 11 son las entradas de la señal analógica de tensión (0 - 10 V), los terminales 4, 7, 10 y 13, las entradas analógicas de corriente. Las salidas de tensión corresponden a los números 14 y 17, y las de corriente al 16 y 19. El terminal 1 es el positivo de la alimentación, 24 V y el 20 la masa. Los terminales 3, 6, 9,12, 15 y 18 son el común de entradas y salidas que tienen que conectarse a masa (terminal 20). El esquema de la figura es el que aparece en la tarjeta de E/S analógicas.

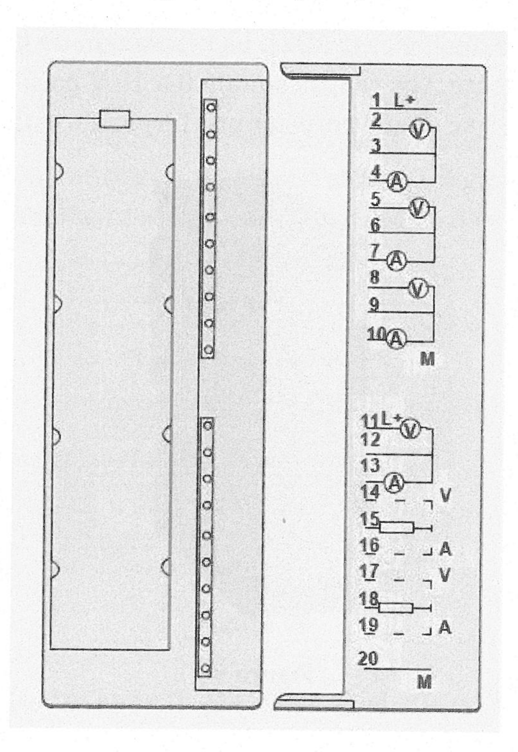


Figura 96.

Si falta la conexión de masa entre M_{ANA} y M, se desconecta la tarjeta y se lee $7FFF_H$ (desbordamiento) en las entradas. Por tanto, es obligatorio que esté unida la masa de las señales analógicas M_{ANA} y la masa de la CPU o de la IM. La masa M_{ANA} (contactos 15 o 18) está en la zona de contactos de salida de la tarjeta.

Advertencia:

La conexión debe ser lo más corta posible y hay que utilizar un cable con una sección de 1 mm^2 como mínimo.

Para los autómatas de la serie S7 1500 se va a utilizar de ejemplo la tarjeta 6ES7531-7NF00-0AB0 para entradas analógicas y la 6ES7532-5NB00-0AB0 para salidas analógicas. En la Figura 97 se representa la tarjeta de entradas analógicas. El aspecto externo de las dos es similar.

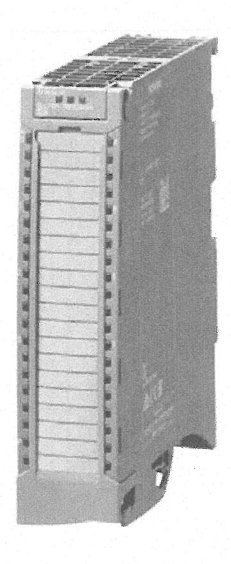


Figura 97

ENTRADAS ANALÓGICAS 1500

La tarjeta 6ES7531-7NF00-0AB0 tiene 8 canales de entradas analógicas seleccionables en tensión o corriente. La resolución es de 16 bits. El módulo tiene las siguientes características técnicas:

- ✓ 8 entradas analógicas con aislamiento galvánico.
- ✓ Tipo de medición: tensión ajustable canal por canal.
- ✓ Tipo de medición: intensidad ajustable canal por canal.
- ✓ Dos modos de operación. Rápido: mínimo periodo de integración 2,5 ms – Estándar: mínimo periodo de integración 7,5 ms.
- ✓ Resolución: 16 bits incl. signo.
- ✓ Diagnóstico parametrizable (por canal).
- ✓ Alarma de proceso al rebasar valores límite ajustable canal por canal (dos límites superiores y dos inferiores, respectivamente).

El conexionado para las entradas de tensión es el que se indica en la Figura 98.

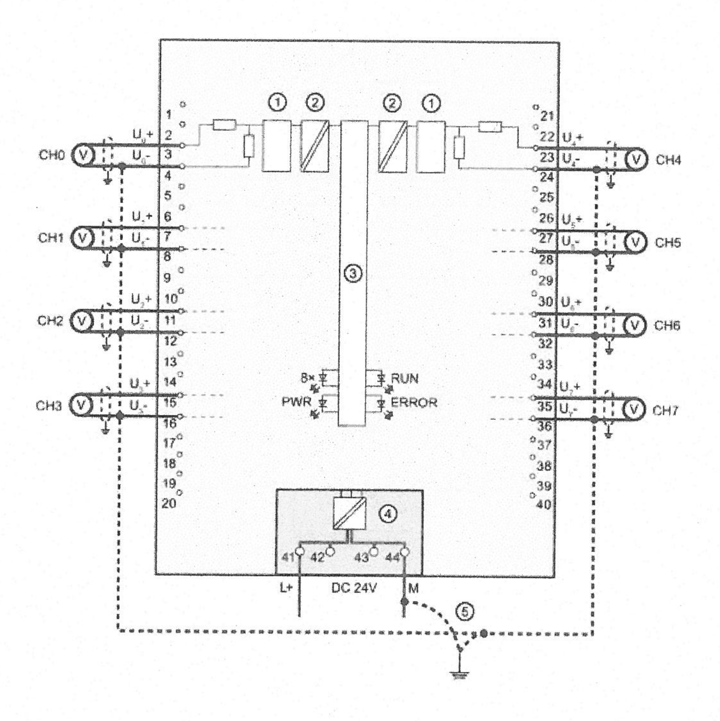


Figura 98

En la siguiente tabla se incluyen los tipos y rangos de medición.

Tipo de medición	Rango de medición
Tensión	de 1 a 5 V ±2,5 V ±5 V ±10 V
Intensidad TM2H (transductor de medida a 2 hilos)	de 4 a 20 mA
Intensidad TM4H (transductor de medida a 4 hilos)	de 4 a 20 mA de 0 a 20 mA ±20 mA

SALIDAS ANALÓGICAS 1500

La tarjeta 6ES7532-5NB00-0AB0 es un módulo de 2 canales de salida digital. El módulo tiene las siguientes características técnicas:

- ✓ 2 salidas analógicas.
- ✓ Resolución: 16 bits incl. Signo.
- ✓ Selección de salida de tensión canal por canal.
- ✓ Selección de salida de intensidad canal por canal.
- ✓ Diagnóstico parametrizable (por canal).

El conexionado para salida en tensión se realiza como se indica en la Figura 99.

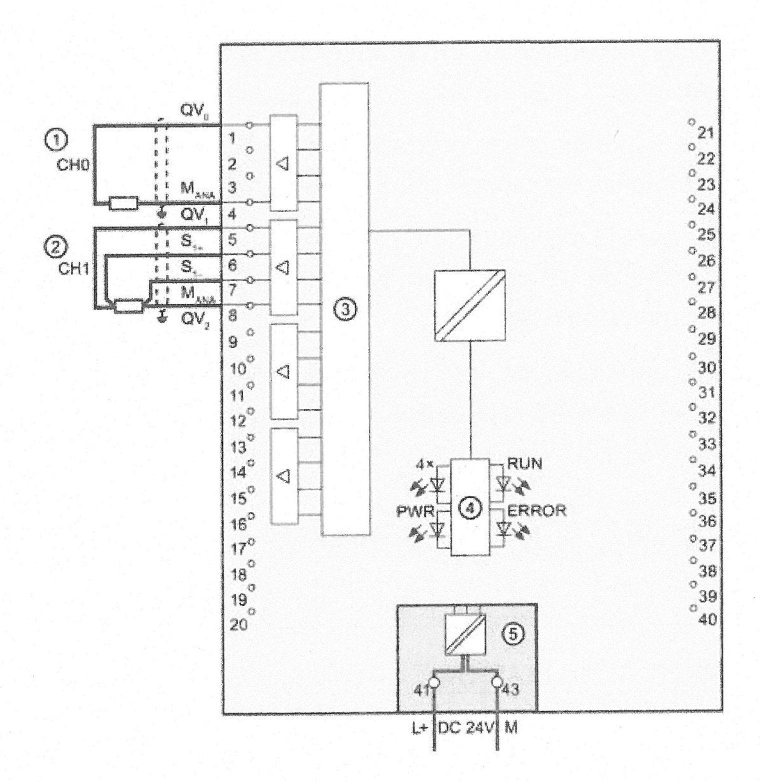


Figura 99

En la siguiente tabla se indica los tipos y rangos de salida.

Tipo de salida	Rango de salida
Tensión	de 1 a 5 V, 0 a 10 V, ±10 V
Intensidad	de 4 a 20 mA
Intensidad	de 4 a 20 mA de 0 a 20 mA ±20 mA

UTILIZACIÓN DE LAS TARJETAS DE ENTRADA Y SALIDAS ANALÓGICAS

CONCEPTOS PREVIOS

En este apartado se va a explicar cómo se utilizan las tarjetas analógicas en los autómatas. Para el S7-300 se utilizará la tarjeta que se ha estudiado en el apartado anterior, SM 334 AI4/AO2, Ref. 334 0CE01 0AA0. Mucho de lo que vamos a ver aquí es válido para los autómatas de la serie S7 1500.

Como se ha dicho, esta tarjeta es de 8 bits, es decir, que utiliza 8 bits para digitalizar cada uno de los valores analógicos, de tensión o corriente, que hay en su entrada. Cada valor analógico tendrá un valor binario en dicho registro. También se sabe que el margen de tensión que debe tener la señal analógica es de 0 a 10 V o de 0 a 20 mA.

El primer objetivo al utilizar una tarjeta de entrada y salida es conocer cuál es el valor binario (o equivalente decimal) en que se convertirá el máximo valor de la entrada, 10 V o 20 mA. Una vez conocido ese dato, ya se estará en disposición de calcular cualquier otro nivel de la entrada realizando una simple regla de tres. Con ese dato podemos conseguir que el PLC actúe

de una manera concreta al reconocer dicho nivel de entrada. Esto se realizará mediante instrucciones de comparación.

Para esta tarjeta se va a estudiar la característica del registro de conversión, que también se debe conocer para el uso de las entradas y salidas analógicas.

La tarjeta emplea 8 bits para la digitalización, pero en realidad el registro de digitalización que utiliza el PLC es de 16 bits, de los cuales utiliza solo 8 para digitalizar y, como veremos, no todos para realizar la conversión a digital.

En la figura se aprecia qué bits, de los 16 que tiene la palabra binaria, emplea para la digitalización, donde BD son los bits utilizados para la digitalización y B los bits de la palabra de todo el registro.

	BD7	BD6	BD5	BD4	BD3	BD2	BD1	BD0							
B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

BITS DE DIGITALIZACIÓN

Como el registro para digitalizar es de 16 bits, cuando se tenga que utilizar en un programa, se deberá tratar como un dato de palabra binaria (W).

Se ha mencionado que el objetivo principal es conocer el valor del registro (palabra de digitalización) cuando la entrada tiene 10 V o 20 mA. Este dato no es fácil de localizar en los manuales, pero sí es muy fácil de conocerlo en la práctica. El valor del registro de digitalización cuando la entrada está a **10 V** es **27648** en decimal, o 0110110000000000 en binario. En la figura se distingue la palabra de digitalización para 10 V.

0	1	1	0	1		1	0	0	0	0	0	0	0	0	0	0

Una vez conocido que para 10 V el valor del registro es 27648, se puede calcular cualquier otra tensión. Por ejemplo, si la tensión de la señal analógica fuera 3 V, el valor de la palabra digital sería:

10 v --------------- 27648

3 v ---------------- X   X= 8294,4

Aproximando, se tomará 8294. Este valor será con el que se debe comparar en el programa, de forma que cuando se detecte el valor 8294, será porque en la entrada hay 3 V. Esos 3 V corresponderán a un valor de la magnitud que se esté tratando: temperatura, velocidad, etc.

Aún se puede analizar un poco más la palabra de digitalización.

El valor mínimo (siguiente al cero) de la palabra de digitalización es la que se muestra en la figura:

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

El valor decimal al que corresponde es $2^7 = 128$, por lo que el valor de este registro tomará valores con escalones de 128 en 128.

Si se desea conocer el valor de tensión que corresponde a cada paquete de digitalización o, dicho de otra forma, cada cuántos voltios se digitaliza la señal, lo podemos determinar con los 8 bits de digitalización. Su valor máximo es (valor digital para los 10 v):

2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	1	0	1	1	0	0	0

Equivale al valor decimal 216, por lo que los 10 V se trocean en......... 10/216 = 46,2 mV. Hay 216 tozos de señal de 46,2 mV cada uno.

CÓMO PROGRAMAR LAS E/S ANALÓGICAS

Solo queda saber cómo se deben utilizar las E/S analógicas en un programa. Para ello, y como con cualquier otra entrada, se necesitan conocer sus direcciones. Estas habrán sido adjudicadas al colocarlas en el *hardware* en Step 7.

En la Figura 100 se muestra un ejemplo de una configuración de un autómata donde se puede observar el direccionamiento de las E/S. Las direcciones son:

Para las estradas: 288...295

Para las salidas: 288...291

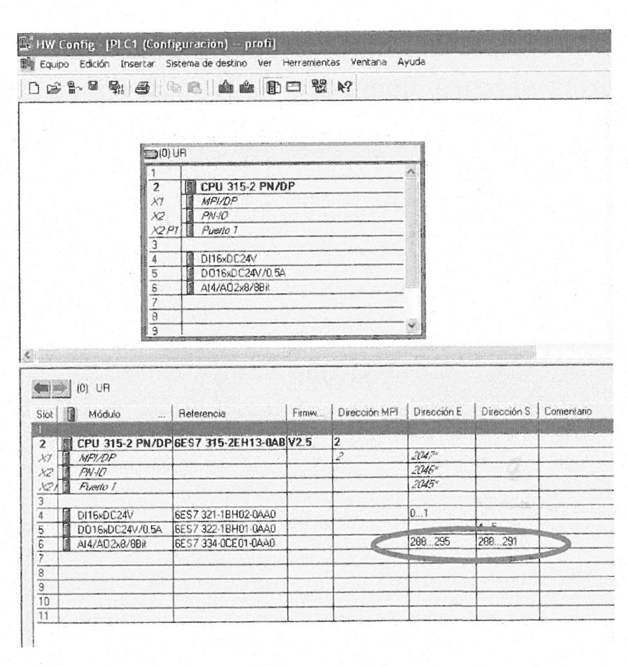


Figura 100

Hay que saber que las **direcciones** de las entradas y salidas **utilizan una palabra**, 16 bits. Por lo tanto, usan dos bytes para su dirección. Entonces, y según lo dicho, las direcciones son:

1.ª dirección de entrada emplea los bytes 288 y 289 // 2.ª dirección de entrada emplea los bytes 290 y 291

3.ª dirección de entrada emplea los bytes 292 y 293 // 4.ª dirección de entrada emplea los bytes 294 y 295

1.ª dirección de salida emplea los bytes 288 y 289 // 2.ª dirección de salida emplea los bytes 290 y 291

Se debe recordar que para nombrar direcciones o datos de palabra (W), o bien doble palabra (DW), debemos usar el primer byte.

Si se desea leer una entrada analógica, se debe emplear la instrucción de carga L. Con ello se lee el valor analógico y se guarda en el ACU1.

Es muy importante saber que los valores de E/S analógicos no pasan por las memorias internas (PAE y PAA) como las digitales. Por lo tanto, hay que leer directamente del pin. Para ello existe la orden leer de periferia que es PE, o escribir en periferia, PA. Para indicar que se lee o escribe de periferia directamente se añade una P a la entrada o salida (no válido para variables tipo bit).

Ejemplo: L PEW288

Esta orden lee el valor analógico de la entrada 288 y 289, y la guarda en el acumulador.

Para utilizar las salidas analógicas se siguen los mismos criterios, pero como es salida (escribir), se debe utilizar la orden transferencia, T.

Ejemplo: T PAW288

Esta orden escribe el valor del ACU1 en la salida analógica 288 y 289.

Es probable que no se conozca el valor máximo de digitalización de nuestra tarjeta, pero se puede determinar de forma práctica. Como uno de los valores de tensión estandarizado es 10 V, se introducen los 10 V por una de las entradas analógicas y luego hay que observar en una tabla de variables el valor que toma el registro de conversión. Como las E/S analógicas no pasan por PAE y PAA, puede que no se vean esos valores en la tabla de variables. Para poder verlo, se puede transferir a una marca de palabra.

```
L PEW 288
T MW20
```

PROGRAMAS DE ENTRADAS Y SALIDAS ANALÓGICAS

Se van a realizar una serie de programas en los que se utilizan las entradas y salidas analógicas. Si se dispone de un entrenador del tipo del que realiza Siemens, se dispondrá de un símil de transductor de una entrada analógica y una salida analógica. Como sensor se

dispone de un potenciómetro que varía su resistencia para obtener una salida en tensión de 0 a 10 V. Está conectado a una entrada digital. Como salida hay un voltímetro.

Igualmente se puede emplear el simulador para realizar estos ejercicios. Se tiene que utilizar una entrada, la PEW288, por ejemplo, y una salida, la PAW 288, y usarla como regulador decimal.

Si se realiza un programa que tome la señal de entrada y la saque por la salida analógica (una simple transferencia de la señal de la entrada a la salida), se verá que, en el caso del entrenador, el voltímetro sigue al valor de tensión en la entrada del potenciómetro. En el simulador se tendrá que desplazar el cursor del regulador decimal con el ratón y se observará que la salida tiene el mismo valor que la entrada.

El programa sería:

```
L PEW 288
T PAW 288
T MW 20
```

T MW 20 → Esta instrucción se incluye solo si se desea ver en una tabla de variables la salida analógica, ya que **no se puede ver.**

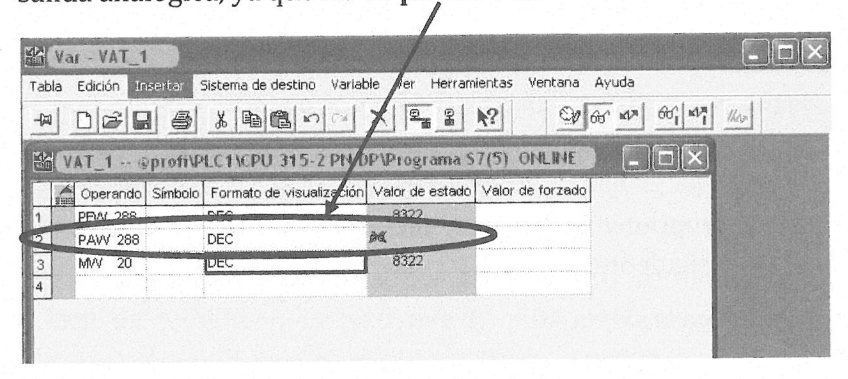


Figura 101

En la Figura 102 se observa el simulador para comprobar el funcionamiento del ejercicio.

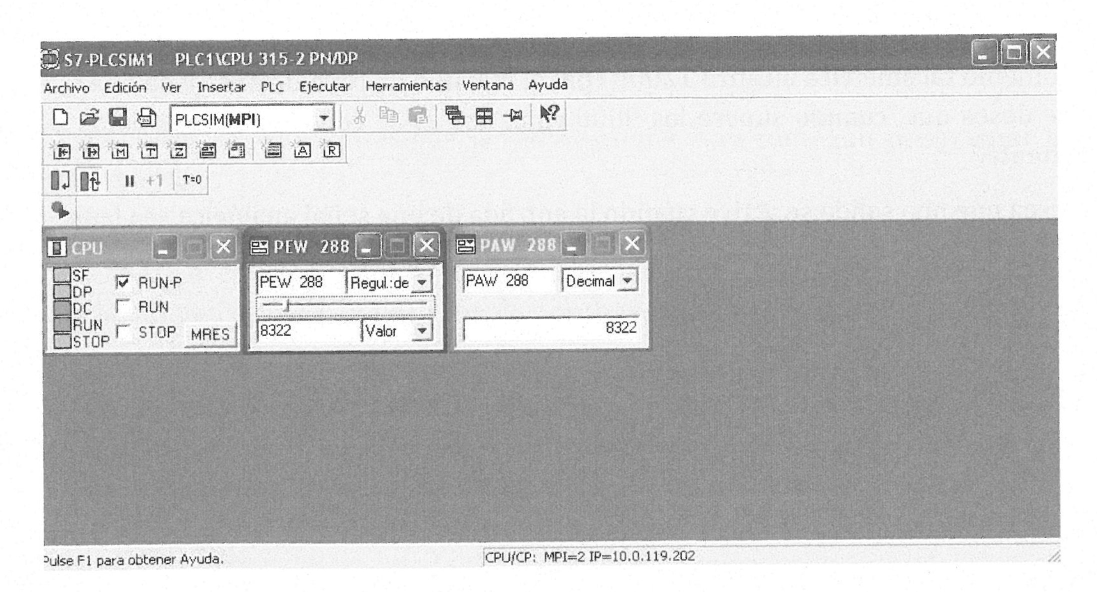


Figura 102

Ejercicio 1

Se desea realizar un programa que active una salida cuando el valor de la tensión analógica supere los 4 voltios.

Lo primero que se debe hacer es calcular el valor del registro de conversión para esos 4 V. Si para 10 V, como se vio en la teoría, era 27648, se calcula la correspondiente regla de tres para conocer el valor binario a los 4 V.

10 / 27648= 4 / x

X= 11059,2   se coge 11059

```
L   PEW 288
T   PAW 288
```

Estas instrucciones se pueden colocar para ver los valores de la entrada y la salida, pero para realizar el programa no es necesario.

```
T   MW  20
L   PEW 288
L   11059
>I
=   A   4.0
```

Cuando el valor del sensor sea 4 V (11.059), se activará la salida.

Ejercicios propuestos

1. Hay que realizar un programa que conecte unas salidas en función del valor de la entrada analógica, según la tabla siguiente:

Si Ve > 2 V, se activa la salida A0.

Si Ve < 5 V, se activa la salida A1.

Si 3 < Ve < 7 V, se activa la salida A2.

Si Ve = 4 V, se activa la salida A3.

(A0, A1, A2 y A3 son direcciones de las salidas del sistema con el que se trabaje).

2. Una dínamo tacométrica alcanza 12000 rpm al máximo de revoluciones y da una tensión de 10 V. Se desea que, cuando supere los 9000 rpm, nos avise conectando una salida de forma intermitente.

3. Se desea que una salida se active cuando la entrada de una señal analógica sea igual a 5 V.

10. PROGRAMACIÓN ESTRUCTURADA: FUNCIONES

INTRODUCCIÓN

La mayoría de las personas que empiezan a programar consideran que hacer un programa es lo mismo que afirmar que el programa «funciona», que cumple con los requisitos planteados en un determinado enunciado. Pero, siendo evidente que eso es importante, no es lo único que interesa. Cuando los programas se van haciendo grandes e importantes, también se deben plantear otros objetivos, además del propio funcionamiento correcto. En este tema vamos a aclarar estas ideas y a aprender a realizar programas «bien hechos y que funcionan».

CONCEPTOS

El primer concepto que se debe aclarar es el significado de «un programa bien hecho», aunque no sea fácil de explicar. La programación estructurada consigue ese objetivo. Simplificando mucho, se puede afirmar que «un programa bien hecho», un programa estructurado, es un **programa bien organizado y claro.** En este capítulo y en el siguiente se explicará cómo organizar bien un programa y para eso utilizaremos ciertos métodos de los que dispone Step 7.

Un programa «bien hecho» no será aquel que tenga 2, 3 o 100 instrucciones menos, si no aquel que sea fácil de modificar, incluso que lo pueda revisar y modificar otra persona diferente a la que lo creó o que, partes de ese programa, puedan reubicarse en otro programa diferente. Cuando se hace un programa hay que pensar en el mantenimiento de las aplicaciones. Un objetivo será la facilidad de depurar errores y la facilidad de introducir mejoras.

ANTECEDENTES. CASO 1

Las primeras formas o estructuras que se van a estudiar para conseguir **programar bien** van a ser las **funciones**.

Pero antes de ver cómo utilizarlas y cuál es su utilidad, se va a realizar un programa sin funciones. Se va a emplear un mismo enunciado en cada una de las estructuras y así se observará la utilidad de cada una de ellas, comprobando las mejoras.

Llevaremos a cabo un programa muy sencillo. La idea de utilizarlo es observar, según se avance, las diferencias de uso de los distintos métodos. El programa tiene poco interés práctico, pero un gran interés en lo referente a obtener los objetivos deseados. Se trata de la creación de un **comparador de dos números de dos bits**. Es evidente que el propio Step 7 tiene ya sus instrucciones de comparación y, por lo tanto, el interés de llevar a la práctica este programa es nulo, pero ya se ha dejado claro cuál es el objetivo real y único del planteamiento.

Siempre que se empieza con el diseño o realización de una idea, se debe pensar cómo llevarla a buen fin. Ahora se va a realizar el comparador de una manera simple, aunque no será la más óptima, según veremos en sucesivos apartados.

El planteamiento es bien sencillo: partimos de dos números A y B de dos bits cada uno, a_1 a_0 y b_1 b_0. Se debe comparar y activar una salida diferente para cada caso siguiendo el siguiente criterio:

A > B, se activa A0.0

A < B, se activa A0.1

A = B, se activa A0.2

Las entradas de los números A y B serán:

E0.0= a_0

E0.1= a_1

E1.0= b_0

E1.1= b_1

Para diseñarlo, primero se creará la tabla de verdad y se simplificarán las funciones obtenidas. Con esas funciones se realizará el programa.

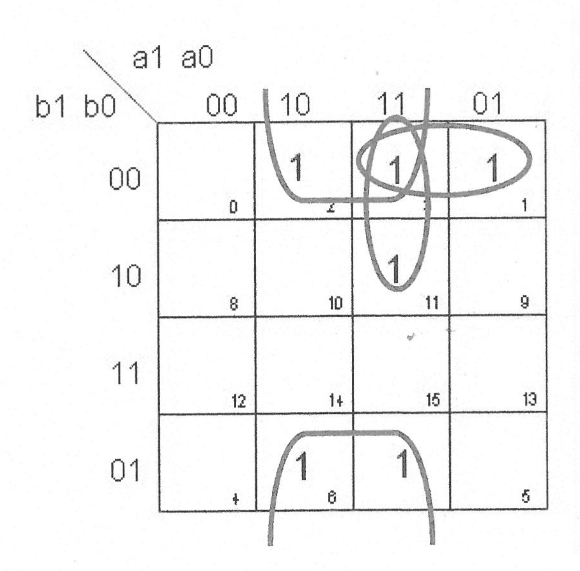
TABLA DE VERDAD

	b_1	b_0	a_1	a_0	A>B	A<B	A=B	Valor A (decimal)	Valor B (decimal)
0	0	0	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0	1	0
2	0	0	1	0	1	0	0	2	0
3	0	0	1	1	1	0	0	3	0
4	0	1	0	0	0	1	0	0	1
5	0	1	0	1	0	0	1	1	1
6	0	1	1	0	1	0	0	2	1
7	0	1	1	1	1	0	0	3	1
8	1	0	0	0	0	1	0	0	2
9	1	0	0	1	0	1	0	1	2

	b_1	b_0	a_1	a_0	A>B	A<B	A=B		Valor A (decimal)	Valor B (decimal)
10	1	0	1	0	0	0	1		2	2
11	1	0	1	1	1	0	0		3	2
12	1	1	0	0	0	1	0		0	3
13	1	1	0	1	0	1	0		1	3
14	1	1	1	0	0	1	0		2	3
15	1	1	1	1	0	0	1		3	3

SIMPLIFICACIÓN POR KARNAUGH

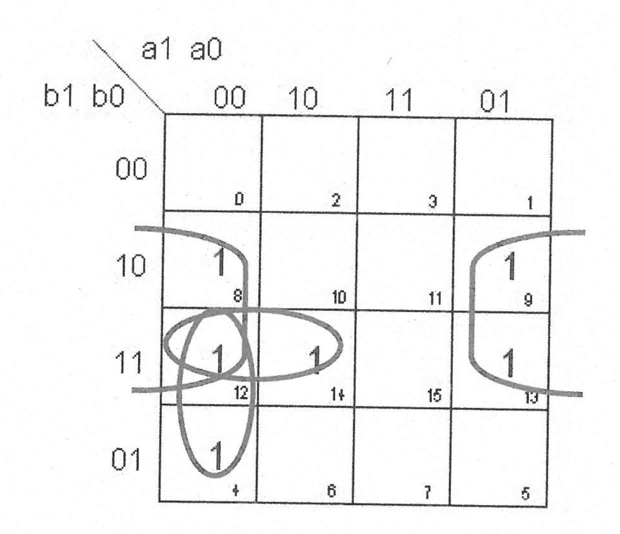
Salida A>B



La función resultante es:

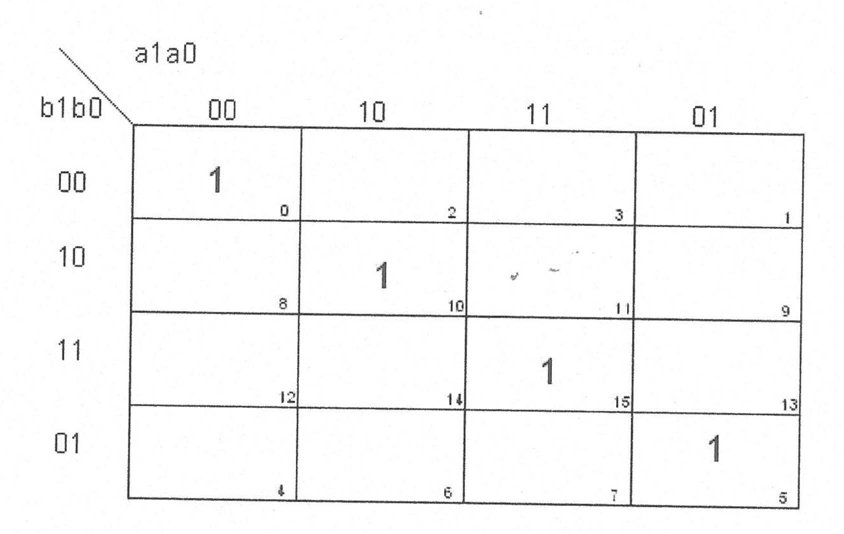
$$A>B = a_0\,\overline{b_0}\,\overline{b_1} + a_0\,a_1\,\overline{b_0} + a_1\,\overline{b_1}$$

<u>Salida A<B</u>



$$\mathbf{A<B= a_0\,\overline{b_0}\,b_1 + a_0\,\overline{a_1}\,\overline{b_0} + a_1\,\overline{b_1}}$$

<u>Salida A=B</u>



En este caso no se puede simplificar.

$$(A=B)= \overline{a_0}\,\overline{a_1}\,\overline{b_0}\,\overline{b_1} + \overline{a_0}\,a_1\,\overline{b_0}\,\overline{b_1} + a_0\,a_1\,b_0\,b_1 + a_0\,\overline{a_1}\,b_0\,\overline{b_1}$$

A continuación, solo queda hacer el programa que consiste en escribir las funciones que han salido. Se han utilizado los símbolos que se indican a continuación:

a0= E0.0 a1= E0.1 b0=E1.0 b1= E1.1

AmyB (A>B)= A0.0 AIB (A=B)= A0.1 AmnB (A<B)= A0.2

Para crear el programa con símbolos es necesario grabar la lista de símbolos antes de programar. Si al introducir un símbolo no lo acepta, se puede introducir anteponiendo al símbolo una almohadilla (#). Es lo que puede ocurrir en este caso con los símbolos a0 y a1. Otra opción, obviamente, es cambiar de símbolo.

Función A>B	Función A<B	Función A=B
U "a0"	UN "a0"	UN "a0"
UN "b0"	U "b0"	UN "a1"
UN "b1"	U "b1"	UN "b0"
O	O	UN "b1"
U "a0"	UN "a0"	O
U "a1"	UN "a1"	UN "a0"
UN "b0"	U "b0"	U "a1"
O	O	UN "b0"
U "a1"	UN “a1”	U "b1"
UN "b1"	U “b1”	O
= "AmyB"	= "AmnB"	U "a0"
		U "a1"
		U "b0"
		U "b1"
		O
		U "a0"
		UN "a1"
		U "b0"
		UN "b1"
		= "AIB"

Se comprueba que el programa es correcto y cumple perfectamente con el enunciado del problema. Pero como ya se ha dicho, no solo se debe pensar en el correcto funcionamiento del programa, sino que hay que tener en cuenta otros factores, por ejemplo, el caso de querer ampliarlo a números de tres, o más, bits. La respuesta es clara, nada de lo realizado serviría. Se tendría que partir de cero y rehacer todo el programa. Se debe concluir que este método no es muy válido, aunque sí cumple con el planteamiento inicial.

CASO 2

Se va a plantear el problema bajo otro prisma, pensando en futuras ampliaciones y mejoras. Lo ideal sería hacer en un programa para **dos números de un solo bit** e ir añadiendo instrucciones si lo que se desea es ampliar más bits. Pero la parte de código realizado debe servir. Ya no se partiría desde cero.

El diagrama de la Figura 103 representa la solución que se acaba de plantear. Se ha realizado para dos bits como establecía el enunciado inicial.

Cada bloque debe tener una entrada para el bit «a» y otra para el bit «b», una salida para el caso de que «a» sea mayor que «b» (a>b), otra para cuando «a» sea menor que «b» (a<b) y otra salida para el caso de igualdad (a=b). Por cada bit que se quiera añadir a los números a

comparar, se añadirá un bloque. Como es lógico, se debe empezar a comparar por los bits de mayor peso. Cada bloque tiene que realizar la comparación siempre y cuando en el bloque anterior (es decir en los bits comparados en el bloque anterior) no haya dado que los bits comparados sean a>b o a <b, en cuyo caso ya se habrá determinado quién es mayor o menor. Los bloques que **no comparan**, porque en algún bloque anterior se resolvió la comparación, debe tener **todas sus salidas a 0**. Como se ha mencionado, la comparación solo la realizará aquel bloque que reciba la señal de igualdad del bloque anterior. Es por este motivo por lo que cada bloque debe tener una entrada de igualdad que se conectará a la salida del bloque anterior y así se tendrá referencia de lo sucedido.

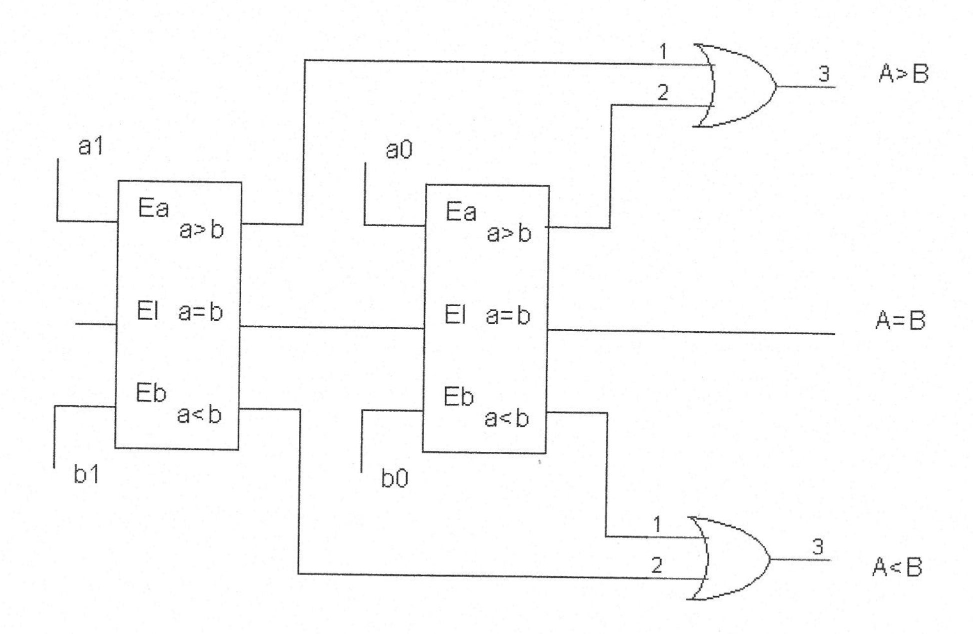


Figura 103

Para aclarar este concepto, se van a incluir algunos ejemplos en decimal. Sean los números A = 452 y B = 422. Si se tuvieran que comparar esos dos números, se debería empezar a comparar por el de mayor valor ponderal, es decir, en este caso por las centenas. Los dos números tienen las centenas iguales (4 y 4), así que todavía no se ha resuelto la comparación. El siguiente bloque recibe una señal de igualdad de los números anteriores y debe comparar. Se determina que el número A tiene un 5 y el B un 2. Ya se ha determinado la comparación, independientemente de las unidades, porque el número A>B y, por lo tanto, todas las salidas del bloque de las unidades serán cero.

Otro ejemplo en decimal: A = 131 y B = 131. Cada bloque tendrá su salida de igualdad a uno, para indicar que en el bloque anterior han sido iguales los bits comparados. Al final, con las unidades, pasará lo mismo y el bit a = b (unidades) determinará que los números A y B son iguales, activando la salida correspondiente.

Esto mismo es igualmente válido para el caso binario.

Una vez visto el concepto del nuevo diseño, se deben resolver las funciones lógicas de cada salida para un bloque. Todos los bloques son iguales.

TABLA DE VERDAD DE UN BLOQUE Y SUS FUNCIONES

Para poder completar la tabla correctamente es necesario recordar que cada bloque debe tener su entrada de igualdad activada (a uno) para que pueda comparar. Si recibe un cero, querrá decir que se ha decidido la comparación anteriormente y, por lo tanto, este bloque no compara y **todas sus salidas serán cero.**

	EI	Eb	Ea	a>b	a<b	a=b
0	0	0	0	0	0	0
1	0	0	1	0	0	0
2	0	1	0	0	0	0
3	0	1	1	0	0	0
4	1	0	0	0	0	1
5	1	0	1	1	0	0
6	1	1	0	0	1	0
7	1	1	1	0	0	1

Ahora se deben sacar las tres funciones de las salidas. Estas son:

$$a>b = EI\,\overline{Eb}\,Ea$$

$$a<b = EI\,Eb\,\overline{Ea}$$

$$(a=b) = EI\,\overline{Eb}\,\overline{Ea} + EI\,Eb\,Ea = EI\,(\overline{Eb}\,\overline{Ea} + Eb\,Ea) = EI\,(\overline{Eb \oplus Ea})$$

Como se aprecia en la función de igualdad, se incluye la función **NOR EXCLUSIVA**. En Step 7 no existe la orden Nor Exclusiva pero sí la Or Exclusiva, por lo que para hacer la XNOR se usará la XOR y luego se negará con la orden NOT. La instrucción XOR (Or Exclusiva) es X.

Ejemplo: $\overline{a \oplus b}$ ⟶

```
U a
X b
NOT
= A0.0
```

EL PROGRAMA

En la siguiente figura se pueden apreciar las entradas y salidas utilizadas en el programa, así como las marcas empleadas en las salidas parciales de cada bloque. El primer bloque debe tener su entrada de igualdad (EI) conectada a uno porque, de lo contrario, según lo explicado anteriormente y al no tener ningún bloque detrás, no compararía (Figura 104).

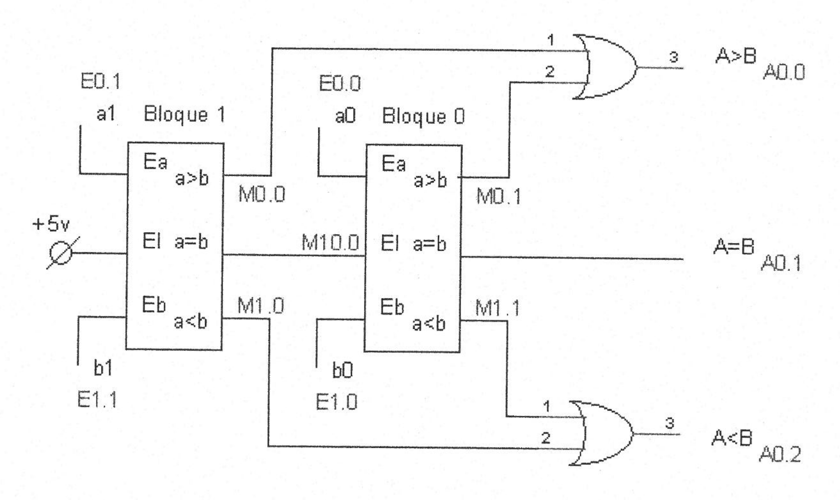


Figura 104

Los símbolos que se han utilizado en el programa son:

Programa S7(5) (Símbolos) -- profi\PLC1\CPU 315-2 PN/DP

	Estado	Símbolo	Dirección	Tipo de dato	Comentario
1		Ea1	E 0.1	BOOL	Entrada a del bloque 1
2	Seleccionar tabla		E 1.1	BOOL	Entrada b del bloque 1
3		Ea0	E 0.0	BOOL	Entrada a del bloque 0
4		Eb0	E 1.0	BOOL	Entrada b del bloque 0
5		EI0	M 10.0	BOOL	Entrada de igualdad bloque 0. La del bloque 1 deber ser uno.
6		AIB	A 0.1	BOOL	
7		AmnB	A 0.2	BOOL	
8		AmyB	A 0.0	BOOL	
9					

Figura 105

El programa es el que se indica en la siguiente tabla. Todas las instrucciones se escribirán en el OB1.

SALIDAS BLOQUE 1			SALIDAS BLOQUE 2		
a>b	a<b	a=b	a>b	a<b	a=b
UN "Eb1" U "Ea1" = M 0.0	U "Eb1" UN "Ea1" = M1.0	U "Ea1" X "Eb1" NOT = "EI0"	U "EI0" UN "Eb0" U "Ea0" = M0.1	U "EI0" UN "Ea0" U "Eb0" = M1.1	U "EI0" U(U "Ea0" X "Eb0" NOT) = "AIB"

SALIDAS FINALES		
A>B	A<B	A=B
U M0.0	U M1.0	
O M0.1	O M1.1	A0.1
= "AmyB"	= "AmnB"	

Una vez realizado este nuevo programa, se observa que funciona igual que el anterior. Son evidentes las ventajas de este método con respecto al anterior:

Si ahora se quiere ampliar a 8 bits, todo lo realizado es válido y no partimos de 0. Este modo es más modular.

FUNCIONES

Vamos a tratar la primera de las estructuras que permiten crear programas «bien hechos» y que no solo «funcionen bien». Ya se ha visto que una de las primeras condiciones para organizar bien un programa es «pensarlo bien», tarea no siempre fácil.

DEFINICIÓN DE FUNCIÓN

Una función es un subprograma que se ejecuta cuando se solicita desde cualquier otro bloque. Cuando una instrucción llama a ejecutar una función, se abandona el programa principal (OB1 u otro) y se pasa el control de programa a la función. Una vez concluida la ejecución de todas las instrucciones de la función, se retorna al lugar del que ha partido. Al contrario que el OB1, que se ejecuta siempre y cíclicamente, las funciones solo se ejecutan cuando se les ordena desde el OB1 u otro bloque. Una función tiene las mismas características que el OB1, excluida la ejecución cíclica. Al bloque función en Step 7 se le denomina FC y se le añade un número: FC1, FC2, etc. También se puede crear un símbolo a la función. En TIA PORTAL es lo mismo.

Una característica de las funciones es que cuando se terminan de ejecutar, el control del programa siempre vuelve a la posición del OB1 (u otro bloque) del que ha partido.

En el siguiente diagrama se observa cómo desde varios puntos diferentes del OB1 se llama a ejecutar la función y cuando termina de ejecutarse se vuelve al punto del OB1 desde donde se ha partido.

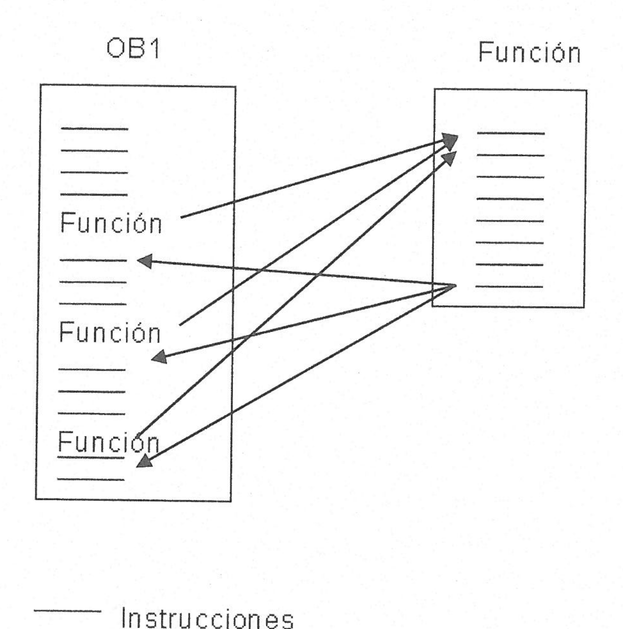


Figura 106

APLICACIÓN DE LAS FUNCIONES

Y ¿para qué sirve una función? O dicho de otro modo, ¿cuándo es necesario utilizar las funciones?

Una función deberá utilizarse cuando en un programa haya bloques de instrucciones que se repiten de una manera continuada a lo largo de todo el programa. Es en ese momento cuando se debe crear un programa independiente en una función y llamarlo cuando sea necesario ejecutar ese grupo de órdenes. Y esto no solo se hace por ahorrar instrucciones, sino por **organizar** el programa, y hacerlo más **modular y** más estructurado.

También se consigue que estas funciones puedan ser útiles en otros programas con el consiguiente ahorro de tiempo en el diseño. De este modo se pueden **reubicar** las funciones en otros programas diferentes.

INSTRUCCIONES PARA UTILIZAR UNA FUNCIÓN

Hay diversas órdenes para utilizar una función en Step 7. Las instrucciones para que se abandone el programa principal, OB1, y se ejecute la función son:

Call FCn   CC FCn   UC FCn

Las tres órdenes abandonan el programa (OB1 u otro) y la ejecución pasa a la función correspondiente. La diferencia es que CALL y UC son incondicionales, mientras que CC es condicional, solo se ejecutará la función si el RLO está activado.

Por ejemplo:

```
U E0.0
CC FC1
```

En este caso, solo cuando la entrada E0.0 se active, se ejecutará la función 1. Si fuera:

U E0.0

CALL FC1

La función 1 se ejecutará siempre, independientemente del estado de la entrada E0.0.

Se examinará alguna diferencia más de estas instrucciones en capítulos siguientes.

Para poder utilizar las funciones, lo primero que se debe hacer es incluirlas en el administrador en bloques, junto al OB1. Para ello, debemos hacer clic en la parte derecha con el botón derecho y seleccionar **Función**.

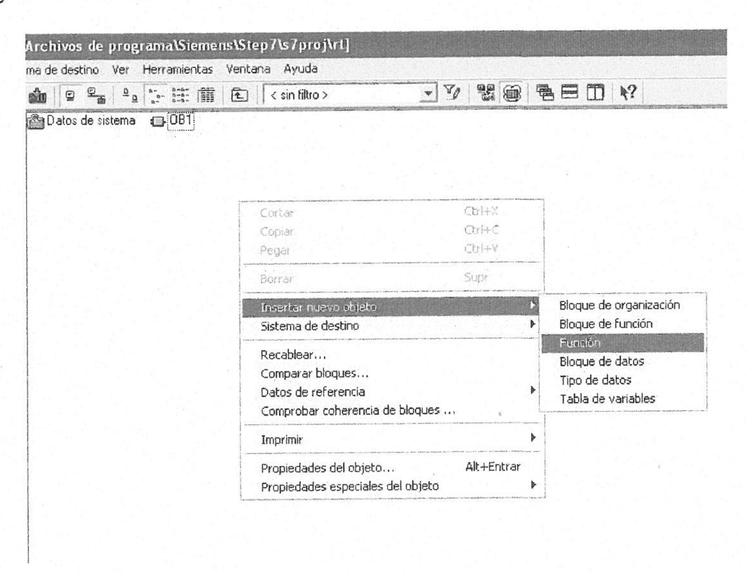


Figura 107

En la Figura 108 se indica el nombre y se acepta. A continuación, se abre la función y se escribe el programa igual que en el OB1.

Propiedades - Función

General - 1ª parte | General - 2ª parte | Llamadas | Atributos

Nombre: FC1

Nombre simbólico:

Comentario del símbolo:

Lenguaje: AWL

Ruta del proyecto:

Ubicación del proyecto: C:\Archivos de programa\Siemens\Step7\s7proj\rt

Código Interface

Fecha de creación: 07/03/2012 08:44:00

Última modificación: 07/03/2012 08:44:00 07/03/2012 08:44:00

Comentario:

Aceptar Cancelar Ayuda

Figura 108

En TIA PORTAL (V13) las funciones se añaden desde **Agregar nuevo bloque,** en **Bloques de programa**, dentro del árbol del proyecto, tal como se indica en la Figura 109. Allí se elige el lenguaje para programar, que puede ser distinto al del OB1. También se puede asignar un nombre distinto al que fija por defecto. Esto es válido para los autómatas de la serie S7 300.

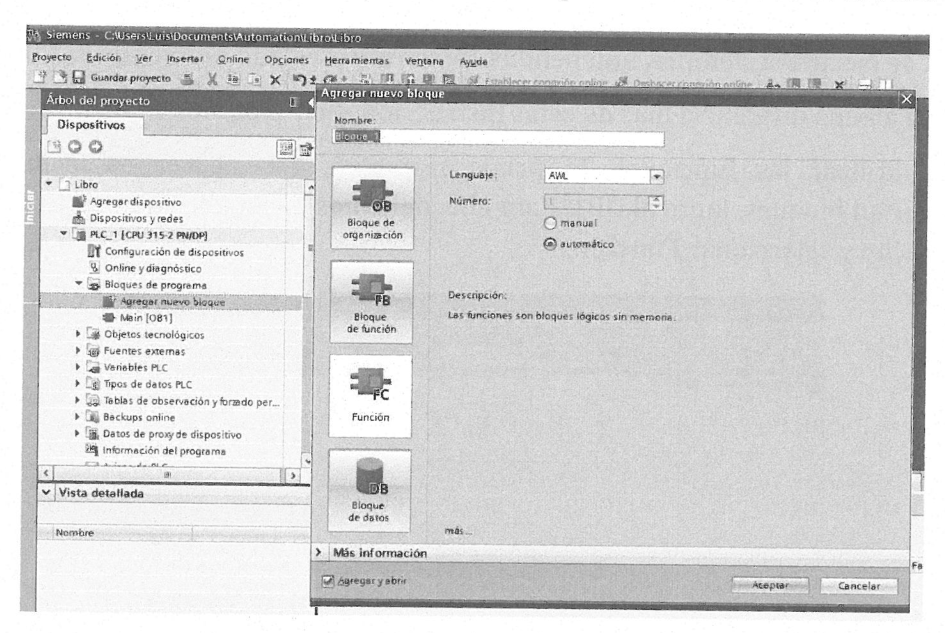


Figura 109

Para el PLC de la serie S7 1500 hay alguna pequeña diferencia que se aclarará en el apartado del PLC 1500.

Mostramos a continuación un ejemplo de cómo usar las funciones. Para ello se va a crear una función que activará una salida con un pulsador cuando se le de paso mediante otra entrada.

En el OB1, y mediante la entrada E0.0, se llamará a la función. Desde la función se podrá activar la salida con la entrada E0.1.

OB1	**FC1**
U E0.0	U E0.1
CC FC1	= A0.0

Para que esto pueda funcionar es necesario que tanto el OB1 como la FC1 estén cargadas en el PLC. Si se olvida de cargar la FC1, el autómata dará un error fatal (led SF encendido).

En este programa solo cuando se **mantenga activada** la entrada E0.0 podrá activarse la salida A0.0, es decir, solo desde la FC1. Hay que tener en cuenta que, solo cuando la función se ejecute, se podrá maniobrar sobre la salida A0.0.

Si una vez dentro se mantiene la entrada E0.1 como interruptor (activada de forma permanente) y, a continuación, se desactiva la entrada E0.0, se puede observar que, por mucho que se quiera apagar la salida, no se podrá conseguir, ya que no se entra en la función. Esto que en este ejemplo es tan sencillo, en ocasiones da muchos problemas si no se tiene el concepto de función muy claro.

EL EJEMPLO DEL COMPARADOR Y LAS FUNCIONES. CASO 3

De nuevo se retoma el ejemplo del comparador. Si se revisa el segundo caso del programa del comparador, se puede observar que hay un grupo de órdenes que se repiten continuamente. De hecho, cada uno de los bloques de comparación individuales son el mismo programa, lo único que cambia es que son direcciones diferentes, tanto de entradas y salidas como de marcas. Sin embargo, las órdenes son las mismas.

Si se pudiera conseguir un único programa, e independiente de las direcciones de cada bloque, es decir, que fuera válido para cualquier dirección, sería un programa perfecto para hacer una función.

En la siguiente figura se observa que lo único que se ha modificado con respecto a lo realizado anteriormente han sido las direcciones, que ahora son otras marcas donde dejar los valores actuales de cada uno de los bloques. Con estas direcciones se creará la función. Cada vez que se desee ejecutar un bloque de comparación (es decir, realizar la comparación de dos bits), se llamará a la función.

Para poder hacerlo correctamente, se tendrá que meter en las marcas que utiliza la función como entradas los valores correspondientes para cada bloque. Las salidas de esta función, que son marcas, se tendrán que guardar para cada uno de los bloques de comparación correspondientes. Si no se hiciera de este modo, se perderían esos datos en la ejecución del siguiente bloque de comparación, ya que es la misma función para cada uno de los bits a comparar.

FUNCIÓN: la Figura 110 representa de forma gráfica la función. Deberá haber tantas llamadas a esta función como número de bits formen los números a comparar.

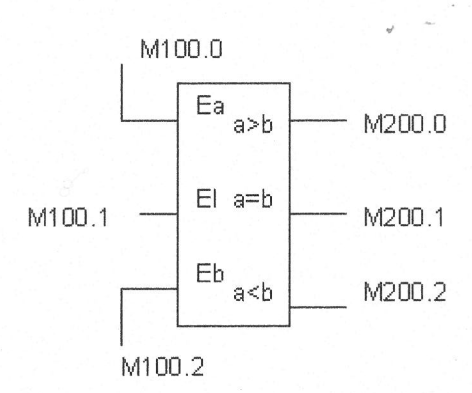


Figura 110

La Figura 111 representa de forma gráfica la nueva situación. Dentro de cada uno de los bloques de comparación ahora se llama a una función, que, como se aprecia, es siempre el mismo bloque. Se indican las marcas en las que se guardarán los datos intermedios de cada una de las comparaciones.

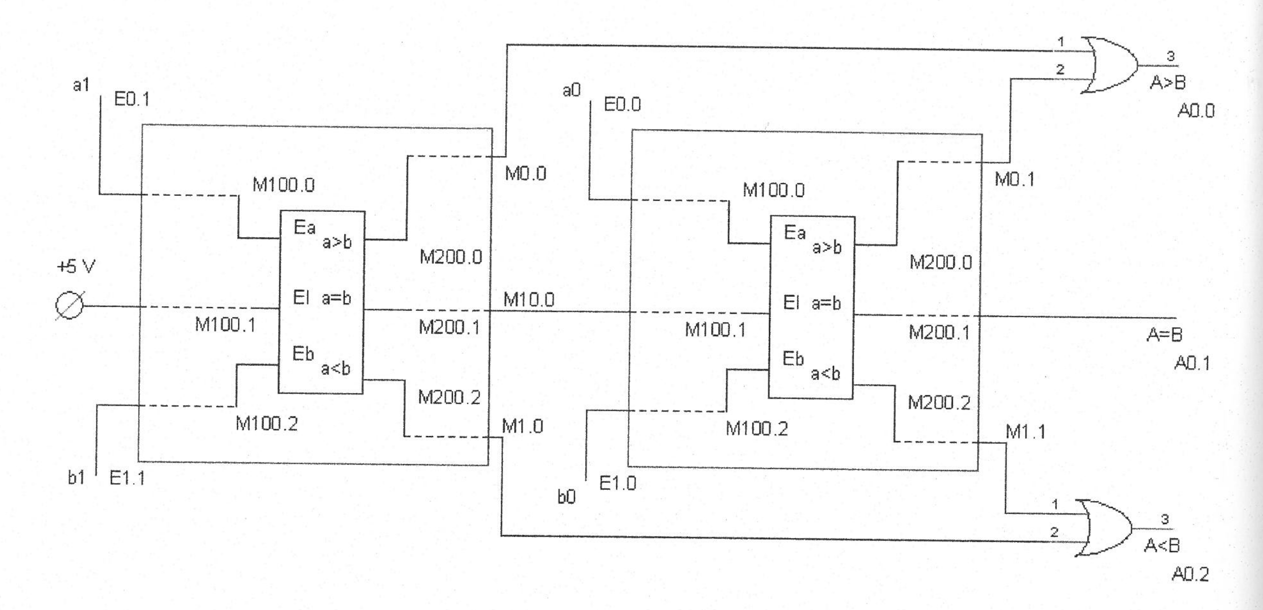


Figura 111

En base a este diagrama se realizará el programa. Se puede observar que, con respecto a la forma de proceder en el Caso 2, hay una gran modularidad y una mejor organización del programa. Además, se deja la posibilidad de poder ampliar sin realizar grandes cambios y utilizando parte del código empleado.

PROGRAMA OB1

```
// BLOQUE 1

// PREPARA LOS DATOS DE ENTRADA PARA LA FUNCIÓN

      U     "Ea1"                  E0.1          -- Entrada a del bloque 1
      =     M     100.0

      SET
      =     M     100.1

      U     "Eb1"                  E1.1          -- Entrada b del bloque 1
      =     M     100.2

      CALL  "FUNCION1"             FC1

// RECOGE LOS DATOS DE SALIDA DE LA FUNCIÓN Y SE GUARDAN EN LAS MARCAS DETERMINADAS.

      U     M     200.0
      =     M       0.0

      U     M     200.1
      =     "EI0"                  M10.0         -- Entrada de igualdad bloque 0. La del bloque 1 deber ser uno.

      U     M     200.2
      =     M       1.0
```

```
// BLOQUE 0

// PREPARA LOS DATOS DE ENTRADA PARA LA FUNCIÓN

      U     "Ea0"                     E0.0          -- Entrada a del bloque 0
      =     M    100.0

      U     "EI0"                     M10.0         -- Entrada de igualdad bloque 0. La del bloque 1 deber ser uno.
      =     M    100.1

      U     "Eb0"                     E1.0          -- Entrada b del bloque 0
      =     M    100.2

      CALL  "FUNCION1"                FC1

// RECOGE LOS DATOS DE SALIDA DE LA FUNCIÓN Y SE GUARDAN EN LAS MARCAS DETERMINADAS.

      U     M    200.0
      =     M      0.1

      U     M    200.1
      =     "AIB"                     A0.1

      U     M    200.2
      =     M      1.1

// OPERACIONES OR

      U     M      0.0
      O     M      0.1
      =     "AmyB"                    A0.0

      U     M      1.0
      O     M      1.1
      =     "AmnB"                    A0.2
```

PROGRAMA FC1

```
U     M     100.0          // Salida a>b
UN    M     100.2
U     M     100.1
=     M     200.0

UN    M     100.0          // Salida a<b
U     M     100.2
U     M     100.1
=     M     200.2

U     M     100.1          // Salida a=b
U(
U     M     100.0
X     M     100.2
NOT
)
=     M     200.1
```

En el supuesto caso de querer aumentar la cantidad de bits de los números que se comparan, solo se debería modificar el programa del OB1 incrementando las órdenes que pasan los datos a la función. La función FC1 no se modificaría.

FUNCIONES CON PARÁMETROS

En el apartado anterior hemos visto que el programa propiamente dicho estaba en la función. El OB1 tan solo se dedicaba a pasar los datos (parámetros) a la función y recoger los que entregaba la función.

El STEP 7 V5.5 y TIA PORTAL tienen otra forma de manejar las funciones en las que haya que pasar parámetros, como es el caso del comparador, aunque no siempre es necesario pasar parámetros a una función.

Vamos a analizar cómo utilizar esta opción en STEP 7 y aplicarla al ejemplo de la comparación. Se verá que es lo mismo, pero con una mayor comodidad a la hora de implementar el programa.

INSTRUCCIONES PARA FUNCIONES CON PARÁMETROS

Hemos visto ya las tres órdenes para llamar a las funciones: CC, UC y CALL. Se dijo que la diferencia era que CC se utiliza condicionada al estado del RLO, mientras que CALL y UC son incondicionales. Sin embargo, hay otra característica que las distingue. Las instrucciones CC y UC no se pueden utilizar con funciones que tengan parámetros en sus propiedades. La sentencia CALL sí puede utilizarse con parámetros.

Ahora vamos a añadir parámetros a las propiedades de una función. En el ejemplo anterior se han pasado parámetros, pero no se han utilizado en las propiedades de la función. Se puede decir que los parámetros se han pasado "a mano" y se han dejado en marcas. Ahora se verá como colocar los parámetros en la propia función. Lo único que cambia con respecto a lo realizado anteriormente es el procedimiento, todo lo demás es lo mismo. El método es más cómodo y lo hace todavía más modular y, por tanto, en caso de una posible modificación futura, más flexible.

APLICACIÓN DE LAS FUNCIONES CON PARÁMETROS AL EJEMPLO DEL COMPARADOR

Cuando se abre la función en la parte superior, aparece lo siguiente (si no se muestra, hay que separar o expandir con el ratón la línea situada debajo de la barra de menús). En la Figura 112 se observa la función con la interfaz de parámetros.

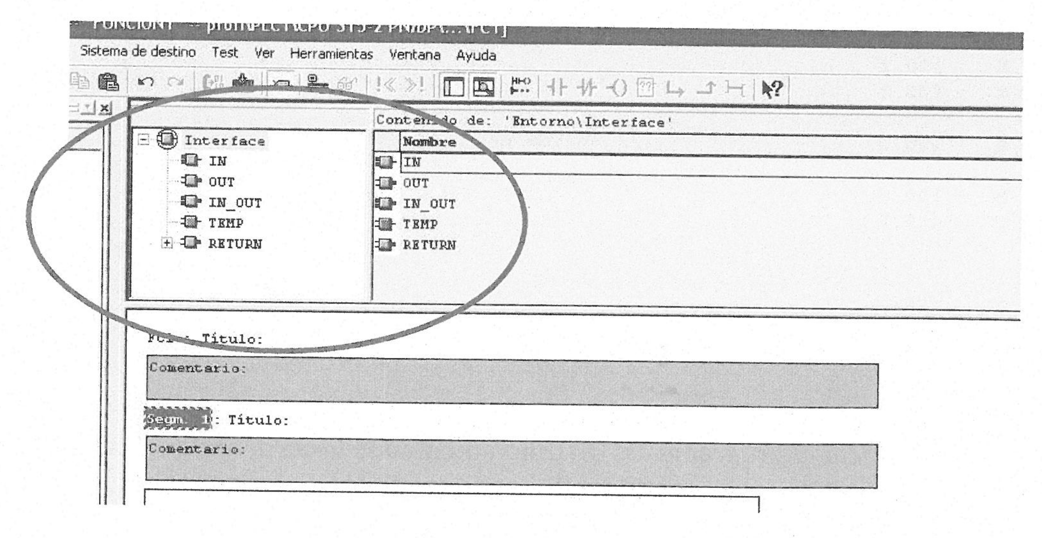


Figura 112

En **Interface** se podrán añadir los parámetros. Estos podrán ser parámetros de:

Entrada (**IN**): aquellos parámetros que se le van a pasar a la función desde el OB1 u otra función.

Salida (**OUT**): aquellos parámetros que se van a transferir desde la función al OB1 u otra función.

Entrada/Salida (**IN_OUT**): sirven tanto para pasar a la función como para transferir desde la función.

Temporales (**TEMP**): solo se utilizan dentro de la propia función. Son locales a la función.

Retorno (**RETURN**): devuelve un valor en **RET_VAL**.

Se introducen los datos de forma simbólica (solo nombres), se toca sobre **IN** y sobre **OUT** (situados en la columna de la izquierda) y se introducen los parámetros (en la columna de la derecha), tal y como se aprecia en la Figura 113 para las entradas y en la Figura 114 para las salidas. Para el ejemplo del comparador se introducen los tres parámetros de entrada (bit a, bit b y bit de igualdad) y los de salida (salida a>b, salida a<b y salida a=b). Se han elegido los nombres que aparecen en las Figuras 113 y 114. Se elige igualmente el tipo de dato del parámetro definido, en este caso booleano (bit).

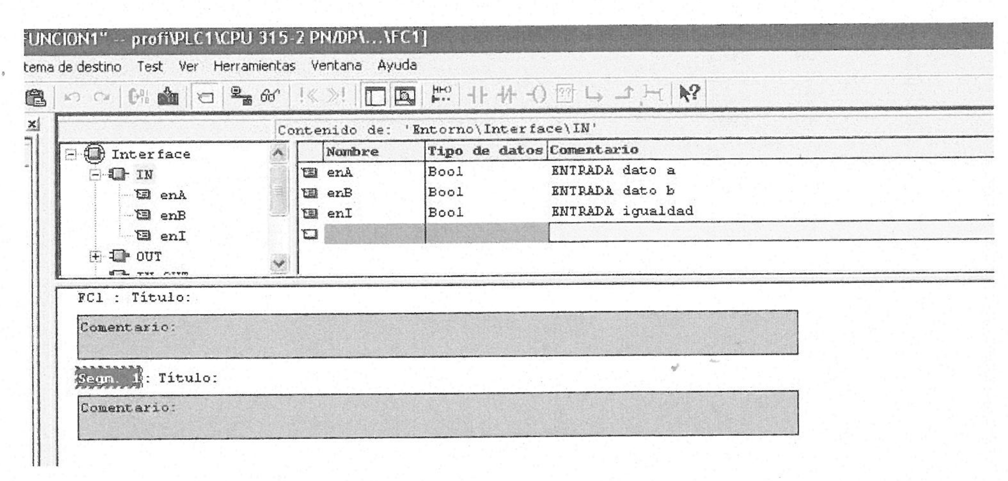


Figura 113

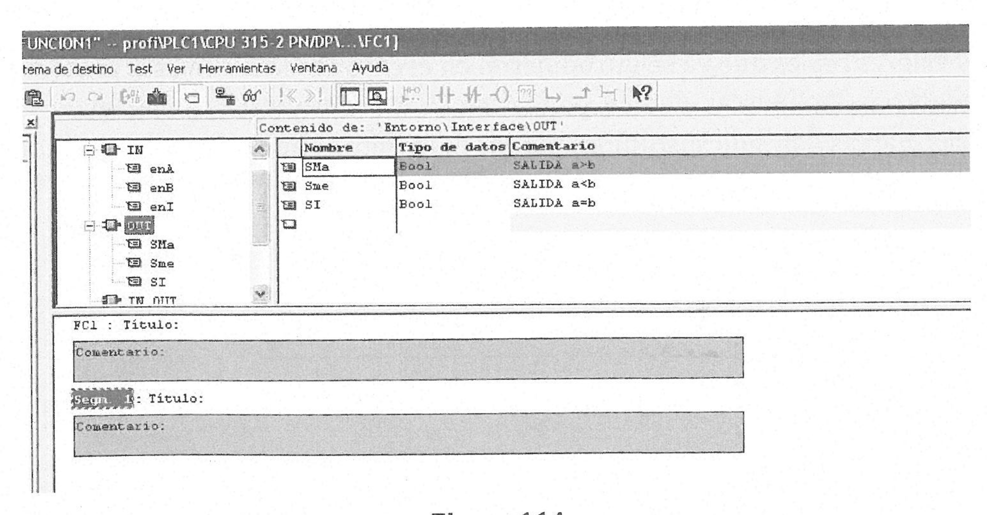


Figura 114

A continuación se escribe el programa poniendo el nombre de los parámetros en vez de los datos de marcas anteriores. Ahora las variables van a ser esos parámetros. Posteriormente en el OB1 se relacionarán a esos parámetros las marcas o salidas reales. La Figura 115 muestra el programa de la función.

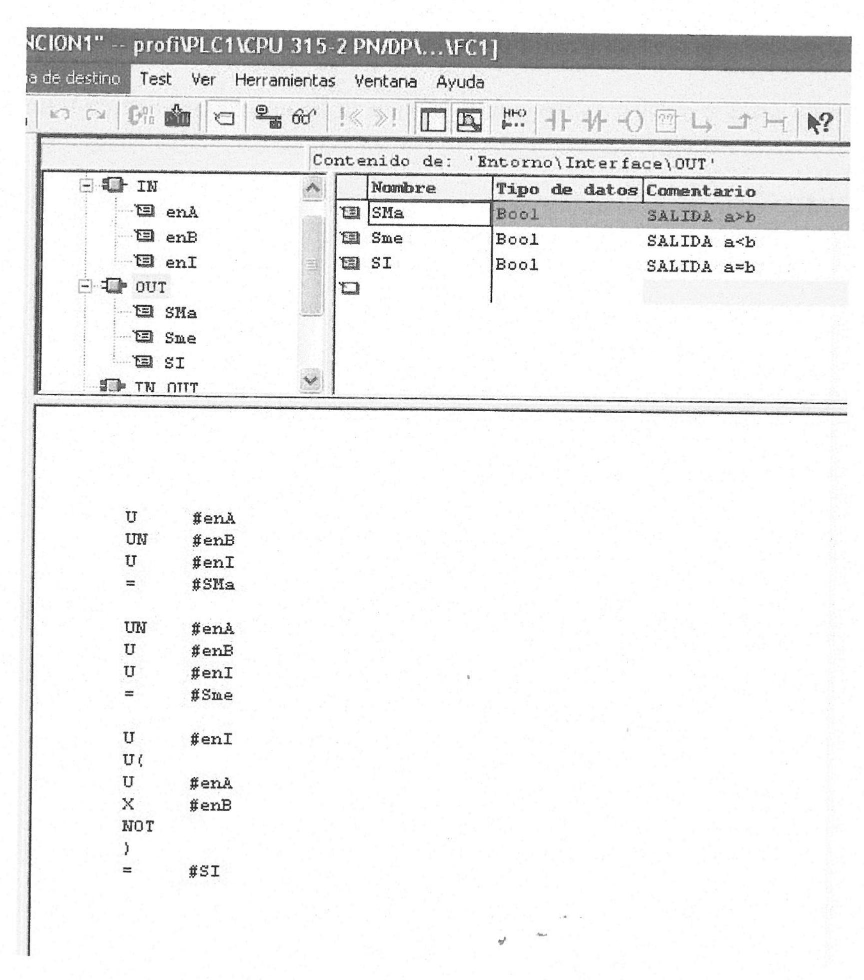


Figura 115

El siguiente paso es escribir el programa OB1. Cuando se coloque la instrucción para llamar a la función FC1, aparecerán en rojo los parámetros definidos en la FC1. Ahí es donde se deben colocar los valores reales que ya se introdujeron en el caso anterior. En este procedimiento ya no son necesarias las marcas que se ponían en la función y que servían para ejecutar la función. Esas marcas han sido sustituidas por los parámetros anteriormente indicados.

PROGRAMA OB1

```
// BLOQUE 1

// PREPARA LOS DATOS DE ENTRADA Y RECOGE LOS DE SALIDA DE LA FUNCIÓN  PARA EL BLOQUE 1

      CALL   "FUNCION1"
       enA:="Ea1"                      // Pasa el valor del bit a1 a la función
       enB:="Eb1"                      // Pasa el valor del bit b1 a la función
       enI:=TRUE                       // Pome a 1 la primera entrada de igualdad
       SMa:=M0.0                       // Guarda en M0.0 la salida a>b
       Sme:=M1.0                       // Guarda en M1.0 la salida a<b
       SI :="EI0"                      // Pasa al bloque siguiente(0) el bit de igualdad

//  PREPARA LOS DATOS DE ENTRADA Y RECOGE LOS DE SALIDA DE LA FUNCIÓN  PARA EL BLOQUE 0
      CALL   "FUNCION1"
       enA:="Ea0"                      // Pasa el valor del bit a0 a la función
       enB:="Eb0"                      // Pasa el valor del bit b0 a la función
       enI:="EI0"                      // Pasa el bit de igualdad del bloque 1
       SMa:=M0.1                       // Guarda en M0.1 la salida a>b
       Sme:=M1.1                       // Guarda en M1.1 la salida a<b
       SI :="AIB"                      //Salida A=B

// OPERACIONES OR

      U      M       0.0
      O      M       0.1
      =      "AmyB"                    // SALIDA A>B

      U      M       1.0
      O      M       1.1
      =      "AmnB"                    // SALIDA A<B
```

Si se compara este programa con el obtenido en los casos anteriores, se ve claramente la modularidad y estructuración conseguida con este último.

EJERCICIOS

1. Se trata de realizar un programa que arranca, mediante un pulsador, tres motores y los mantiene en marcha durante un tiempo determinado. Disponen de tres ciclos diferentes de tiempos. Cada ciclo se activa mediante el estado de un contador que se incrementa o decrementa mediante sendos pulsadores. Se pasará de un ciclo a otro mediante la activación de un pulsador que incremente el contador u otro pulsador que lo decremente. El ciclo no se podrá cambiar mientras los motores no se paren. Otro pulsador parará los motores. Primero se selecciona el ciclo con los contadores y luego se arrancan con el pulsador de marcha.

Se pide hacerlo de dos formas: con funciones sin parámetros y con funciones con parámetros.

Los ciclos y sus tiempos son:

CICLO	MOTOR 1	MOTOR 2	MOTOR 3
1	5 s	8 s	20 s

CICLO	MOTOR 1	MOTOR 2	MOTOR 3
2	7 s	6 s	8 s
3	4 s	12 s	15 s

2. Se trata de realizar el automatismo para una fábrica en la que se elaboran dos tipos de bebidas. Los parámetros que determinan el tipo de bebida son las cantidades de los tres productos que componen cada una de ellas y el tiempo que permanecen mezclándose los tres productos.

La cantidad se determinará según el tiempo en que esté activada la válvula de caída de cada silo de producto. Las bebidas son: Bebida 0 y Bebida 1, y los productos son: Producto A, Producto B y Producto C.

La bebida 0 tiene los siguientes parámetros:

A	B	C	Agitador
3 s	4 s	5 s	6 s

La bebida 1 tiene los siguientes parámetros:

A	B	C	Agitador
2 s	5 s	7 s	4 s

Se dispone de un interruptor general de puesta en servicio de la instalación y de un selector de bebida.

Se debe hacer de dos formas diferentes, con funciones sin parámetros y funciones con parámetros.

11. PROGRAMACIÓN ESTRUCTURADA: BLOQUE DE FUNCIÓN (FB) Y BLOQUE DE DATOS (DB)

INTRODUCCIÓN

En el tema anterior se ha visto cómo, utilizando las funciones, se pueden crear programas mucho más modulares y mejor organizados. En este capítulo se va a seguir en la misma idea: conseguir programas más organizados y estructurados. Para ello se van a utilizar el FB (bloque de función) y los DB.

El concepto de un FB es el mismo que el de un FC, la diferencia es la utilización de un bloque de datos asociado.

Todo lo que se afirme aquí es válido para la plataforma TIA PORTAL, referido siempre a la serie S7 300. Para el PLC S7-1500 casi todo será igual, pero en el apartado del PLC 1500 se verá alguna diferencia a la hora de proceder para llamar a los FB.

Como se ha visto en el tema anterior con las funciones, habrá que ir al árbol del proyecto y agregar un nuevo bloque.

BLOQUE DE DATOS (DB)

Un bloque de datos es un módulo donde se introducen datos, no se escribe código. Es el único módulo utilizado en la programación que no lleva código. Tiene el siguiente aspecto:

DB1 -- "MI_DB1" -- Pruebas DB y FC y FB\PLC1\CPU 315-2 DP\...\DB1

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	DB_VAR	INT	0	Variable provisional
+2.0	dato1	BOOL	FALSE	
+3.0	dat2	BYTE	B#16#0	
+4.0	dato3	BOOL	FALSE	
=6.0		END_STRUCT		

Figura 116

Hay dos tipos de DB: los **globales** y los de **instancia** o locales. Los globales puede utilizarlos cualquier módulo de programación. Los de instancia pertenecen obligatoriamente a un FB.

BLOQUE DE DATOS GLOBALES

Este tipo de DB puede utilizarlo el OB1, las FC y los FB. No pertenece a ninguno de ellos y por eso se denomina bloque de datos globales. Para poder utilizar un bloque de datos global se

debe añadir en bloques para posteriormente introducir los datos. Esto es válido para Step 7 V5.5 y TIA PORTAL.

Para ello, desde el administrador general y desde bloques se añade un **Bloque de datos**, tal como se indica en la Figura 117:

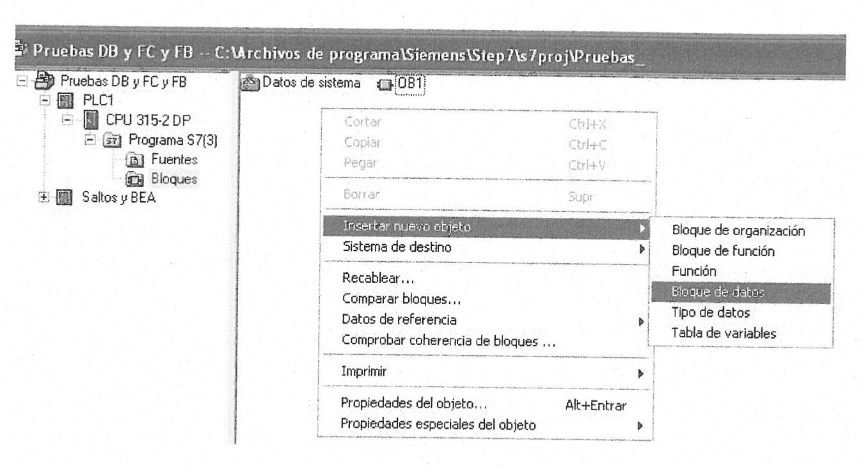


Figura 117

Posteriormente se abre el DB. Para poder introducir/editar los datos, tiene que estar en la vista **Declaración**, según se muestra en la Figura 118.

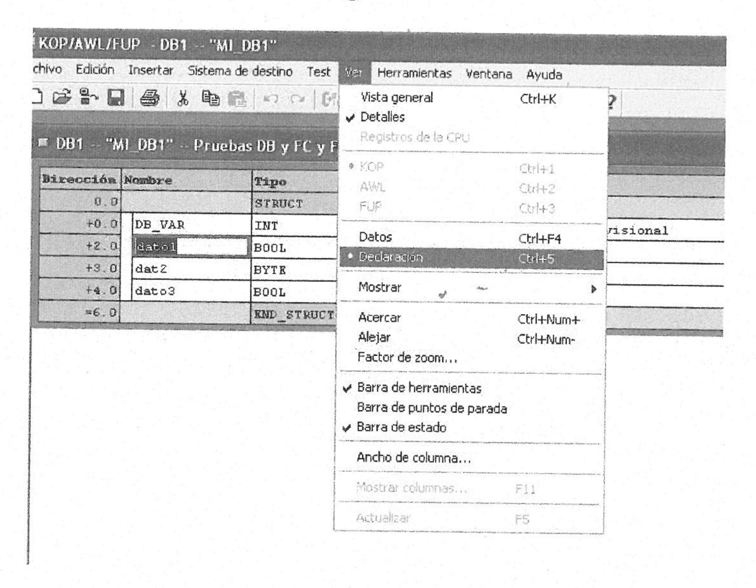


Figura 118

El primer dato (DB_VAR) sale por defecto y se puede borrar si se desea. Para borrar una fila es suficiente con marcarla haciendo clic en **Dirección** y suprimirla.

En la columna **Nombre** se asigna el nombre al dato introducido. En **Tipo** se indica el formato del dato de que se trata. Se selecciona con el botón derecho del ratón y se indica según sean datos simples o compuestos. En **Valor inicial** se incluye el valor que debe disponer al inicio. Si no se indica nada, ese valor será el que tenga por defecto el tipo de dato de la variable utilizada. Por ejemplo, en una marca ese valor inicial, por defecto, es cero. En comentarios se puede colocar una descripción del uso del dato.

El **Campo de dirección** es el que asigna el DB en función del tipo de dato indicado. Esa dirección es la que se debe indicar en el programa para hacer referencia al dato.

Para poder ver y modificar el valor actual hay que acceder en el DB a **Ver/Datos.** Al guardar por primera vez el DB, y si no se indica otra cosa en el valor actual, este pasa a ser el valor inicial. El valor actual solo se puede ver y cambiar desde la vista **Datos**. Los datos no se actualizan en la vista del DB, sino con F5 para ver el valor actual. Si se selecciona **Online** (con las gafas), se pueden ver esos valores.

INSTRUCCIONES PARA UTILIZAR UN BLOQUE DE DATOS

En un programa o módulo de programa se pueden utilizar varios DB. Para hacer uso de cada uno de ellos primero, se debe **abrir el DB** que se desea leer o escribir. La instrucción que se debe usar es AUF. Se utiliza del siguiente modo:

```
AUF DB n
```

Donde n es el número del DB correspondiente. Cada vez que se abre un DB, la lectura o escritura de todos los datos se hace desde el DB abierto. No es necesario cerrar el DB cuando se desea abrir otro. Cuando se abre un DB, se cierra el que esté abierto y NO puede haber más de uno abierto.

Para nombrar los datos, y leerlos o escribir, es necesario conocer la dirección que le ha asignado el DB cuando se ha escrito dicho dato y su tipo. Esto es válido para STEP7 V5.5. Para TIA PORTAL, los datos se nombran según el nombre con que se han escrito en el DB.

Como ya se ha mencionado, lo primero que se debe hacer es abrir el DB. Una vez abierto, para acceder a sus datos se deben nombrar. Si se desea escribir en el DB, sería de la siguiente forma:

```
AUF DB1
U E0.0
= DBX 4.0
```

La X indica que el dato es tipo bit. Si fuera byte, se escribiría B, si fuera palabra, W, y si fuera doble palabra, una D. El estado del bit de entrada E0.0 se escribe en el DB1, concretamente en la posición 4.0, que es (y debe ser) de tipo booleano.

Otro ejemplo, es este caso de lectura del dato:

```
AUF DB1
U DBX 2.0
= A0.0
```

Hay otro modo de leer o escribir en los DB. En esta otra forma no es necesario abrir el DB con la orden AUF. En la propia orden se indica el DB en el que se desea leer o escribir. Aquí se incluyen algunos ejemplos:

Para escribir:

```
U E0.0
= DB1.DBX4.0 (también = DB1.dato3, según Figura 116 )
```

Para leer:

U DB2.DBX2.0 (también U DB1.dato1, según Figura 116)
= A0.0

PROGRAMA DEL COMPARADOR CON FUNCIÓN Y BLOQUE DE DATOS GLOBALES

Se va a seguir con el ejemplo del comparador, pero ahora utilizando una función y un bloque de datos global. El programa será similar al que se hizo con función sin parámetros. La única diferencia es que ahora la función, en vez de utilizar marcas como variables de entrada y salida, utiliza el DB.

Lo primero que se debe hacer es crear el bloque de datos y después realizar la función. De esta forma, cuando se vaya a realizar la función, ya se conocerán las direcciones que ocupan nuestros datos en el DB.

El DB será como el que aparece en la Figura 119.

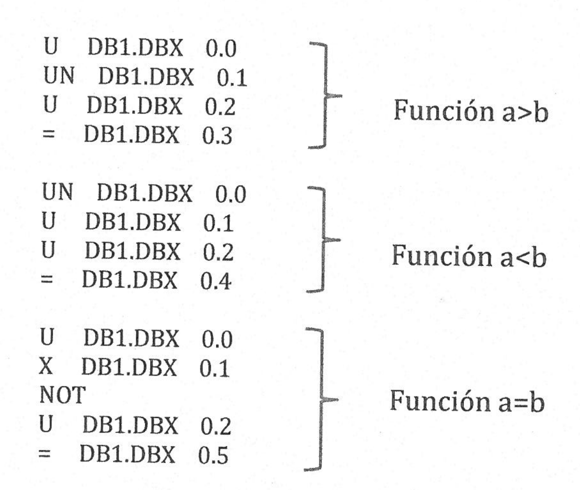
KOP/AWL/FUP - [DB1 -- profi\PLC1\CPU 315-2 PN/DP]

Archivo Edición Insertar Sistema de destino Test Ver Herramientas Ventana Ayuda

Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	Ea	BOOL	FALSE	
+0.1	Eb	BOOL	FALSE	
+0.2	Ei	BOOL	FALSE	
+0.3	SMa	BOOL	FALSE	
+0.4	Sme	BOOL	FALSE	
+0.5	Si	BOOL	FALSE	
=2.0		END_STRUCT		

Figura 119

El programa de la función es:



```
U   DB1.DBX  0.0
UN  DB1.DBX  0.1
U   DB1.DBX  0.2
=   DB1.DBX  0.3
```

Función a>b

```
UN  DB1.DBX  0.0
U   DB1.DBX  0.1
U   DB1.DBX  0.2
=   DB1.DBX  0.4
```

Función a<b

```
U   DB1.DBX  0.0
X   DB1.DBX  0.1
NOT
U   DB1.DBX  0.2
=   DB1.DBX  0.5
```

Función a=b

El OB1 debe preparar los datos antes de llamar a la función y guardar los datos que da la función para después hacer las operaciones OR.

BLOQUE 1

PREPARA LOS DATOS DE ENTRADA Y RECOGE LOS DE SALIDA DE LA FUNCIÓN PARA EL BLOQUE 1:

```
U   "Ea1"
=   DB1.DBX  0.0

U   "Eb1"
=   DB1.DBX  0.1

SET
=   DB1.DBX  0.2

CALL "FUNCION1"

U   DB1.DBX  0.5
=   "EI0"

U   DB1.DBX  0.3
=   M   0.0

U   DB1.DBX  0.4
=   M   1.0
```

BLOQUE 0

PREPARA LOS DATOS DE ENTRADA Y RECOGE LOS DE SALIDA DE LA FUNCIÓN PARA EL BLOQUE 0:

```
U   "Ea0"
=   DB1.DBX  0.0

U   "Eb0"
=   DB1.DBX  0.1

U   "EI0"
=   DB1.DBX  0.2

CALL "FUNCION1"

U   DB1.DBX  0.5
=   "AIB"              ——————————>  SALIDA A=B

U   DB1.DBX  0.3
=   M   0.1
```

```
U   DB1.DBX  0.4
=   M   1.1

U   M   0.0
O   M   0.1
=   "AmyB"          ————————>  SALIDA A>B

U   M   1.0
O   M   1.1
=   "AmnB"          ————————>  SALIDA A<B
```

BLOQUE DE FUNCIÓN (FB)

Los FB son módulos de programación similares a las funciones. La diferencia radica en que los FB llevan asociado un DB, por lo menos. A estos DB, que pertenecen al FB, se denominan de instancia y solo pueden ser utilizados por el FB al que pertenecen. El FB puede tener asociados más de un DB.

El procedimiento de creación de los DB para este caso es diferente. No hay que crearlo previamente. Se crea al llamar al FB desde el OB1, como se comprobará a continuación. Este procedimiento es igual para STEP 7 V5.5 y para TIA PORTAL (V13).

El proceso de realización será el siguiente:

1. Se crea el FB desde **Administrador general**. En **Bloques** y con el botón derecho se debe insertar nuevo objeto. En TIA PORTAL es desde **Agregar nuevo bloque**.
2. Se abre el FB y se crean parámetros como se hizo con las funciones.
3. Se escribe la función con los parámetros creados.
4. Se escribe el OB1. Cuando se llama al FB, se pregunta si se desea crear el DB que no existe y se dice que sí. De este modo se habrá creado el DB con los datos de los parámetros del FB.

Para llamar a un FB, se indica del siguiente modo:

```
CALL  FB1,  DB1
```

Cuando se llama al FB aparecen los parámetros de ese bloque de función, que a su vez son los datos del bloque de datos. Por ejemplo, en esta llamada:

```
CALL  FB1,DB1

enA:=E0.1

enB:=E1.1

enI:=TRUE

SMa:=M0.0

Sme:=M1.0

SI :=M10.0
```

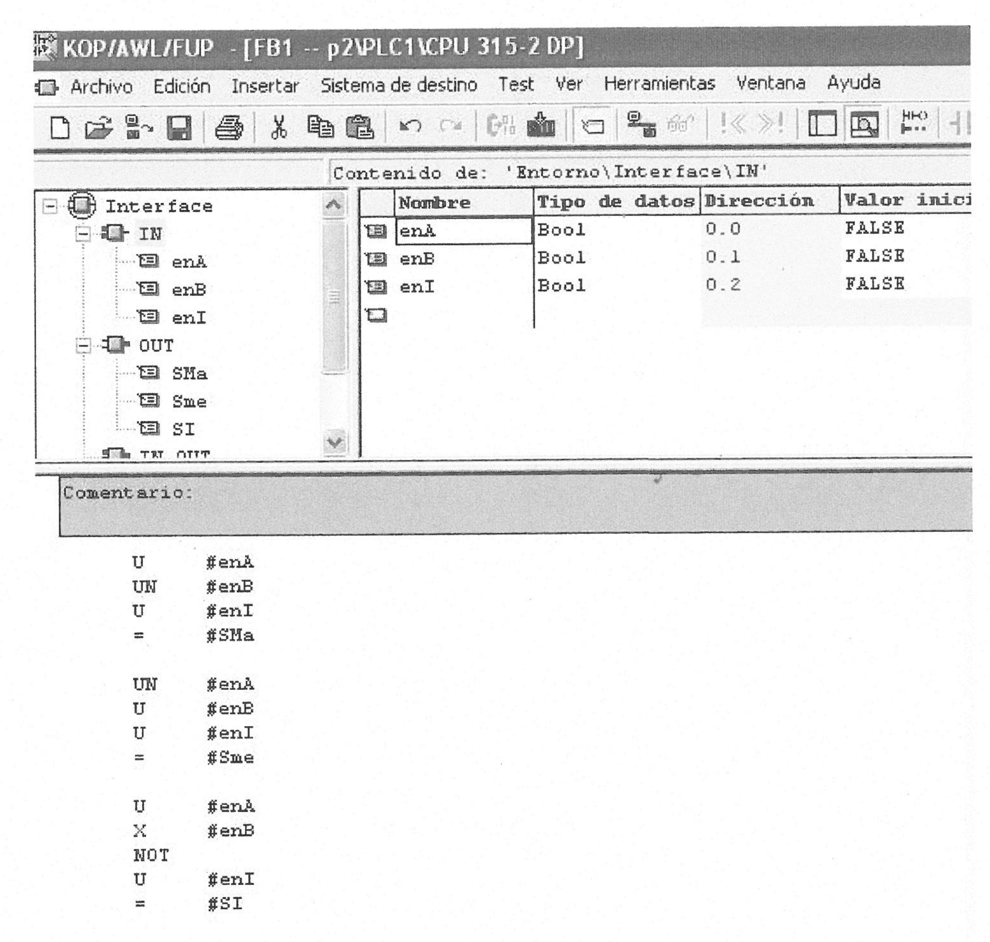
Como se hacía antes con las funciones, se le dan los valores oportunos a los parámetros de entrada y la función deja los valores en los parámetros de salida. La diferencia es que, en este caso, los datos se intercambian en el DB asociado.

De igual modo, es posible no indicar el valor de un parámetro cuando se llama a la función, es decir, se puede dejar en blanco. Así, se coge el valor que tiene ese parámetro en el DB, pero en este caso el valor inicial no se añade en el DB sino en el FB cuando se definen los parámetros. Existe una columna para el valor inicial.

PROGRAMA DEL COMPARADOR CON BLOQUE DE FUNCIÓN (FB1) Y BLOQUE DE DATOS DE INSTANCIA (DB1)

Se va a realizar el programa del comparador utilizando un FB y un DB de instancia.

Primero se crean los parámetros del FB y luego el programa, tal como se detalla en la Figura 120.




```
U    #enA
UN   #enB
U    #enI
=    #SMa

UN   #enA
U    #enB
U    #enI
=    #Sme

U    #enA
X    #enB
NOT
U    #enI
=    #SI
```

Figura 120

Ahora en el OB1, cuando se realice la primera llamada al FB1, se creará el DB1 con los parámetros como datos de ese DB (Figura 121).

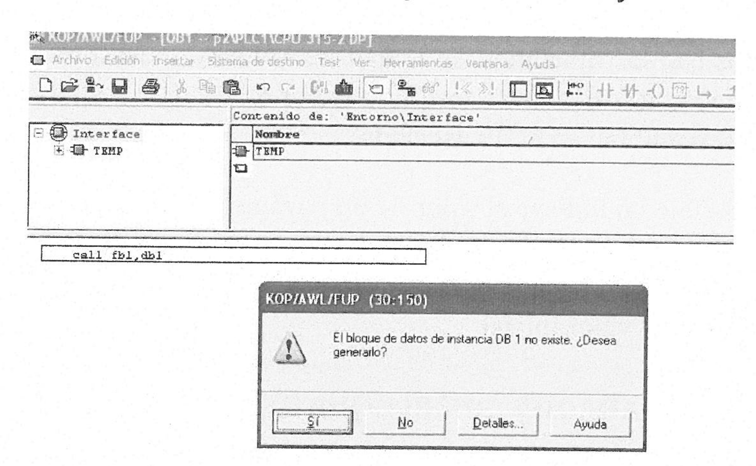


Figura 121

El OB1 es:

```
CALL  FB     1 , DB1
 enA:=E0.1
 enB:=E1.1
 enI:=TRUE
 SMa:=M0.0
 Sme:=M1.0
 SI :=M10.0

CALL  FB     1 , DB1
 enA:=E0.0
 enB:=E1.0
 enI:=M10.0
 SMa:=M0.1
 Sme:=M1.1
 SI :=A0.1

U     M      0.0
O     M      0.1
=     A      0.0

U     M      1.0
O     M      1.1
=     A      0.2
```

Como se puede observar, en el ejemplo del comparador creado con FB y DB no se aprecia una gran mejora en la organización con respecto a los avances conseguidos con las funciones sin y con parámetros. La única diferencia es el uso de un DB para guardar los datos, en vez del uso de marcas. Y este es el procedimiento habitual.

Como ya se ha comentado, en el apartado del PLC S7-1500 se indicará qué hay que hacer para acceder a las funciones y a los bloques de funciones. Tiene que ver con la forma de utilizar la memoria este PLC.

12. GRAFCET

INTRODUCCIÓN

En los temas anteriores hemos explicado cómo realizar un programa con una organización y un desarrollo más modular y estructurado. Pero no se ha estudiado ningún método que ayude a programar. Cuando los programas se hacen grandes, es importante seguir un método, un sistema más o menos estándar.

Grafcet es un método para programar, una ayuda a la programación. Sirve para organizar el trabajo. No es un nuevo lenguaje de programación, de hecho sirve para cualquier lenguaje de programación de PLC, sino un método gráfico que se realiza previamente a la programación. Una vez realizado el grafcet, el siguiente paso es crear el programa.

Este procedimiento que se utilizará ahora solo sirve para aquellos sistemas que sean secuenciales, es decir, que realizan los procesos de una manera cíclica y con operaciones, claramente, una detrás de otra.

Grafcet es el acrónimo de *Graphe de Comands Etape/Transition* (Gráfico Funcional de Control de Etapas y Transiciones).

PROCEDIMIENTO GRAFCET

Lo primero que se tendrá que hacer para realizar un programa utilizando este sistema es dibujar el grafcet y para ello se debe conocer el procedimiento y los elementos que se utilizan. A continuación se deberá trasladar lo indicado en el grafcet a instrucciones del autómata.

PARTES DE UN GRAFCET

Un grafcet está formado por **etapas** y **transiciones**. El grafcet tiene que evolucionar de una etapa a otra cumpliendo con las condiciones de la transición. En la Figura 122 se incluye un ejemplo con las etapas y con las transiciones. Las etapas se representan mediante cuadrados. Dentro de ellos se coloca el número de etapa. Las transiciones se indican con una pequeña línea horizontal.

Para unir las epatas se utiliza una línea vertical. Por defecto, el criterio de evolución del grafcet es siempre de arriba a abajo. No se colocan flechas, excepto cuando resulta necesario para ofrecer mayor claridad.

Las **transiciones** son la condición o grupo de **condiciones** (unidas mediante lógica binaria) que se deben cumplir para que se pueda pasar de una etapa a otra.

En las **etapas** se encuentran todas las **acciones** que deben activarse cuando se active la etapa correspondiente.

Las acciones se colocan a la derecha de la etapa y se encierran en un rectángulo algo más fino. Esto al fin y al cabo es un criterio estético, que no modifica en nada el correcto funcionamiento

del grafcet. Las acciones y condiciones se pueden indicar de forma explícita o bien de forma abreviada. También se puede hacer referencia a las partes del sistema/autómata o no. Igualmente se pueden crear dos grafcet, uno de forma abreviada y más técnica y otro más textual que defina claramente cada acción y condición.

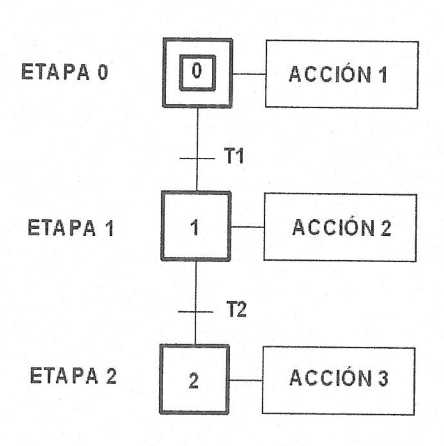


Figura 122

En la Figura 122 se pueden distinguir las distintas transiciones, las etapas y las acciones de cada etapa.

El siguiente ejemplo aclarará la forma de proceder en un grafcet. Se supone que la transición 1 (T1) es un sensor de proximidad inductivo que detecta el paso de un objeto. Este detector está colocado en la entrada E0.1. Para completar el ejemplo se supone que en la acción 2 (en la etapa 1) se debe poner en marcha el motor que hace que una cinta trasportadora se mueva. El contactor que consigue que ese motor gire está conectado a la salida del autómata A0.0.

Según lo enunciado, cuando se esté en la etapa 0 y se cumpla la condición de la transición 1 (T1) (es decir, que el detector inductivo conectado a la entrada E0.1 se active), se debe conectar la cinta (motor, salida A0.0).

En la Figura 123 se aprecian diferentes grafcet para expresar lo mismo, dependiendo de si se hace más o menos explícito.

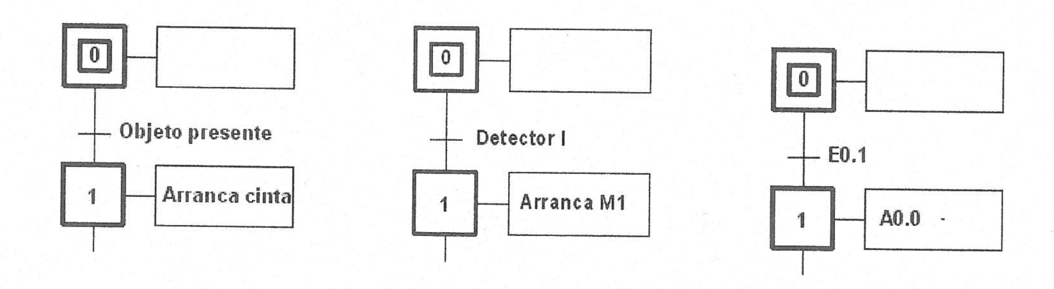


Figura 123

No es determinante la forma de hacerlo e incluso se pueden mezclar alguna de ellas. Parece evidente que la primera es la más clara ya que las indicaciones tienen relación con la función que se desempeñan en el proceso automático. Pero es importante igualmente relacionar el grafcet con las E/S utilizadas. Por eso si, además de indicar las acciones y transiciones con palabras, se indican en una tabla las E/S reales utilizadas en el PLC, el grafcet será mucho más claro. También se pueden añadir en el propio grafcet esas entradas y salidas.

CONDICIONES DE EVOLUCIÓN

¿Qué tiene que suceder para que se pase de una etapa a otra? Por ejemplo, en los casos anteriores qué tiene que suceder para pasar de la etapa 0 a la etapa 1. Ya hemos comentado que debe cumplirse la condición de la transición, pero es evidente que no solo debe suceder eso, porque si el sistema se encuentra en la etapa 3 y se cumple la transición 1, el proceso no se debe modificar. La condición de la transición solo tendrá efectos sobre la evolución si se encuentra en su etapa correspondiente. En este caso solo se pasará a la etapa 1 si se está en la etapa 0 y además se cumple la condición de transición 1.

Resumiendo, para pasar de una etapa a otra es indispensable que se esté en la etapa y se cumpla la condición de transición para pasar a la siguiente.

PROGRAMAR CON UN GRAFCET

Una vez realizado el grafcet, el siguiente paso es escribir el programa a partir de ese grafcet. Hay que ser estrictos a la hora de crear el programa: **primero** se realizará el código para las **transiciones** y **después** para las **acciones**. Se puede hacer también al revés, pero **nunca se deben mezclar** ambos conceptos. El método es muy sistemático, es decir, que siempre se hace igual. Solo cambian las transiciones y las acciones.

Cada una de las etapas se representa con una marca de bit. Es conveniente hacer coincidir la dirección de la marca con el número de la etapa para un mejor seguimiento del programa y del grafcet. También es conveniente que las marcas sean lo más correlativas posibles.

En un graceft lineal nunca debe haber dos etapas activadas simultáneamente, ya que el sistema debe ser secuencial y las etapas que se van activando deben desactivar la anterior.

Para entender la manera en que se debe programar, es conveniente saber cómo evolucionan las diferentes etapas. En el ejemplo siguiente ¿qué debe ocurrir para que se evolucione de la etapa 1 a la 2?

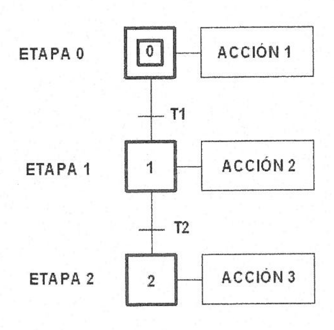


Figura 124

La respuesta correcta sería:

1º Debe estar activada la etapa n.º 1.

2º Debe cumplirse la condición de la transición T2.

Y una vez que se consiguen esas dos condiciones ¿qué debe hacerse para que el grafcet siga su camino secuencial?

1º Desactivar la etapa n.º 1.

2º Activar la etapa n.º 2.

Como cada una de las etapas es una marca, activar o desactivar una etapa es lo mismo que activar o desactivar la marca de esas etapas.

En el siguiente ejemplo completo se va a comprobar el proceso de realización de un grafcet.

Ejercicio 1

Se trata de arrancar tres motores en cascada condicionados a la marcha. Solo hay un motor en marcha simultáneamente: cuando arranca un motor se desconecta el que estaba en marcha. Cada motor tiene un pulsador de arranque. La secuencia es M1-M2-M3. El motor M3 se para con otro pulsador.

El grafcet del automatismo es:

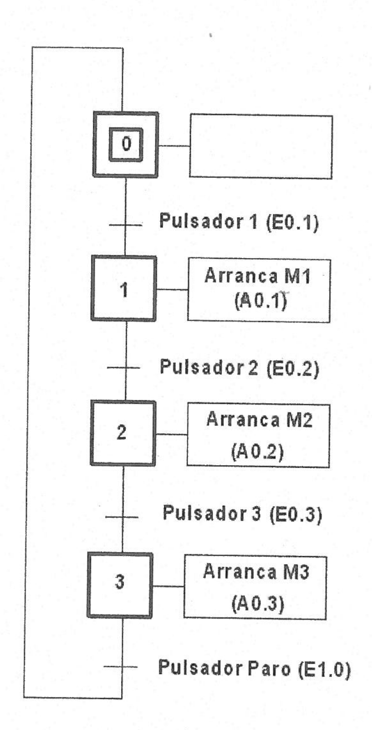


Figura 125

Cada etapa estará representada por una marca. En este caso:

- Etapa 0..... Marca M0.0
- Etapa 1..... Marca M0.1

- Etapa 2..... Marca M0.2
- Etapa 3..... Marca M0.3

La etapa 0 no tiene ninguna acción. Podía pensarse que, al no tener acción, se podría eliminar, pero el hecho de no tener acción no significa que no realice su cometido. En este caso esa etapa 0 hace que la etapa 3 se desactive y, por lo tanto, se desconecte el motor M3 y el sistema quede parado a la espera de volver a activar P1.

Para realizar el programa primero se programan las transiciones y luego las acciones.

TRANSICIONES

Transición Pulsador 1

```
U E0.1
U M0.0
R M0.0
S M0.1
```

Estando en la epata 0 y cumpliéndose la activación de E0.1, se resetea la etapa anterior M0.0 y se pasa a la siguiente, M0.1.

Transición Pulsador 2

```
U E0.2
U M0.1
R M0.1
S M0.2
```

Estando en la epata 1 y cumpliéndose la activación de E0.2, se resetea la etapa anterior M0.1 y se pasa a la siguiente, M0.2.

Transición Pulsador 3

```
U E0.3
U M0.2
R M0.2
S M0.3
```

Estando en la epata 2 y cumpliéndose la activación de E0.3, se resetea la etapa anterior M0.2 y se pasa a la siguiente, M0.3.

Transición Pulsador Paro

```
U E1.0
U M0.3
R M0.3
S M0.0
```

Estando en la epata 3 y cumpliéndose la activación de E1.0, se resetea la etapa anterior M0.3 y se pasa al principio, M0.0.

ACCIONES

ETAPA 1

```
U M0.1
= A0.1
```

Arranca MOTOR1

ETAPA 2

```
U M0.2
= A0.2
```

Arranca MOTOR2

ETAPA 3

```
U M0.3
= A0.3
```

Arranca MOTOR3

En el grafcet anterior y en su programa se observa que no hay que parar los motores, ya que, al activarlos mediante asignación (=), cuando se desactiva su etapa (su marca de etapa), se desactivará también sus acciones.

Una aclaración importante para que este y todos los ejercicios funcionen es que para poder arrancar el grafcet es necesario que, al principio, alguna de sus etapas (marcas) estén activas. Para empezar, es lógico que la primera etapa que debe activarse sea la etapa 0 (marca M0.0), pero esto no puede suceder si no se activa de alguna manera externamente el propio grafcet. Para conseguirlo y hacerlo de forma automática, es decir, que nada más encender el PLC el grafcet arranque entrando en la etapa 0, se debe utilizar el OB 100.

El OB 100 es un bloque de organización **que solo se ejecuta una vez** y únicamente cuando el autómata pasa **de STOP a RUN o RUNP**. Para ello se deberá tocar el conmutador del PLC, pasarlo a STOP y posteriormente a RUN. Si no se hace esto, el OB100 no se ejecutará y el grafcet no funcionará.

Dentro del PLC se puede escribir cualquier programa, ya que es un bloque de programación, igual que el OB1 con la objeción de que solo se ejecuta una única vez. Es lógico que en el OB100 se introduzcan todas aquellas órdenes que sirvan para inicializar nuestros programas.

Para abrir el OB100 lo primero que se debe hacer es insertarlo en bloques junto al OB1, se hace igual que cuando se insertaba el OB1. Con el botón derecho del ratón en la zona de bloques se hace clic en **Insertar nuevo objeto** y se selecciona **Bloque de organización** cambiando el número de OB que salga por 100. Se abre y se escribe el programa. No hay que olvidar enviarlo al PLC como cualquier otro bloque.

En el OB100 se puede escribir esto:

```
SET
S M0.0
R M0.1
R M0.2
R M0.3
```

La orden SET pone a uno el RLO (estado lógico del programa) y activa todas las ordenes que tiene por debajo. Se pone a uno la marca M0.0 para entrar en el grafcet y, por seguridad, aseguramos que al principio todas las demás marcas estén a cero.

Y no se debe olvidar pasar el conmutador del PLC de STOP a RUN.

Es muy útil utilizar una tabla de variables con todas las marcas del grafcet en línea y orden para observar que el grafcet se ejecuta bien (Figura 126). Entre otras opciones, se podrá detectar si en algún momento hay dos etapas activadas simultáneamente.

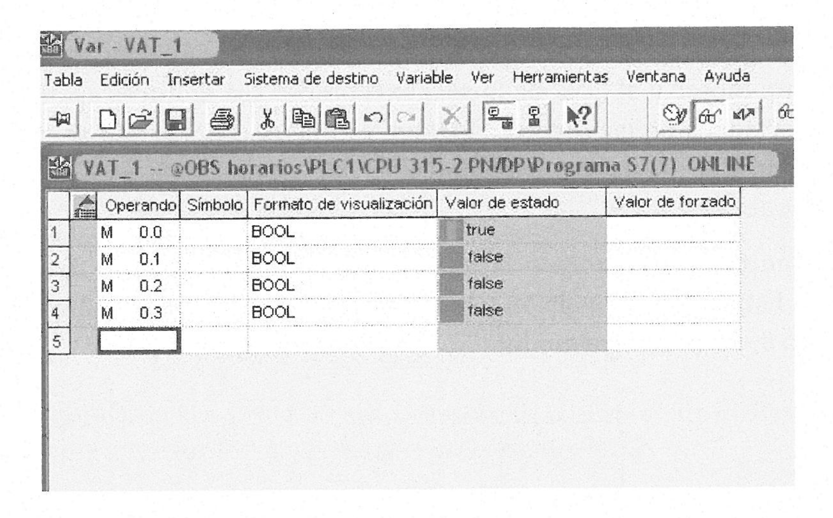


Figura 126

Ejercicio 2

Se trata de arrancar tres motores en cascada condicionados a la marcha. Todos los motores se quedan en marcha y solo se podrán desconectar cuando estén los tres en marcha. El paro será de los tres simultáneamente. Cada motor tiene un pulsador de arranque. La secuencia es M1-M2-M3. Los motores se paran con otro pulsador.

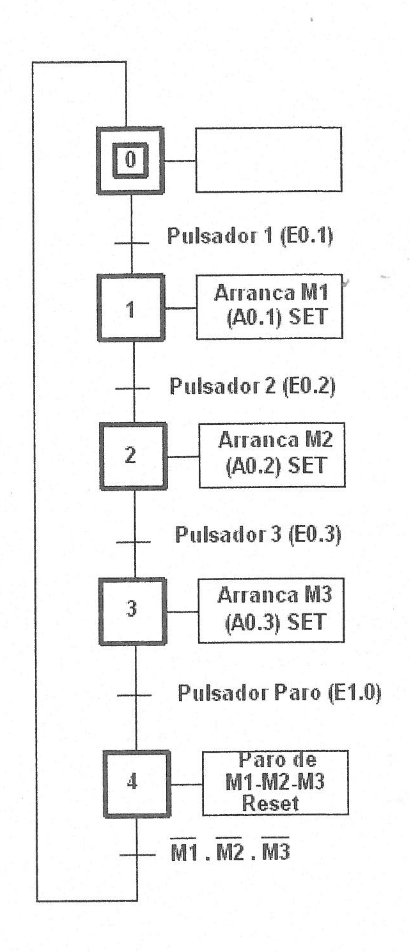


Figura 127

El enunciado es diferente al ejercicio 1. Ahora los motores permanecen arrancados cuando se conecta el siguiente motor y al final de la secuencia todos los motores están en marcha. Esto va a cambiar las acciones de las diversas etapas, ya que ahora, cuando se deba desactivar la etapa que ha arrancado un motor, este no tiene que pararse. De este modo, la acción no seguirá a la propia etapa.

Las transiciones no cambian, excepto la última que es nueva. Cuando todo esté parado, se comienza el ciclo. Esta última transición y la acción podrían evitarse si pasamos el paro de los motores a la etapa 0. Sería de este modo:

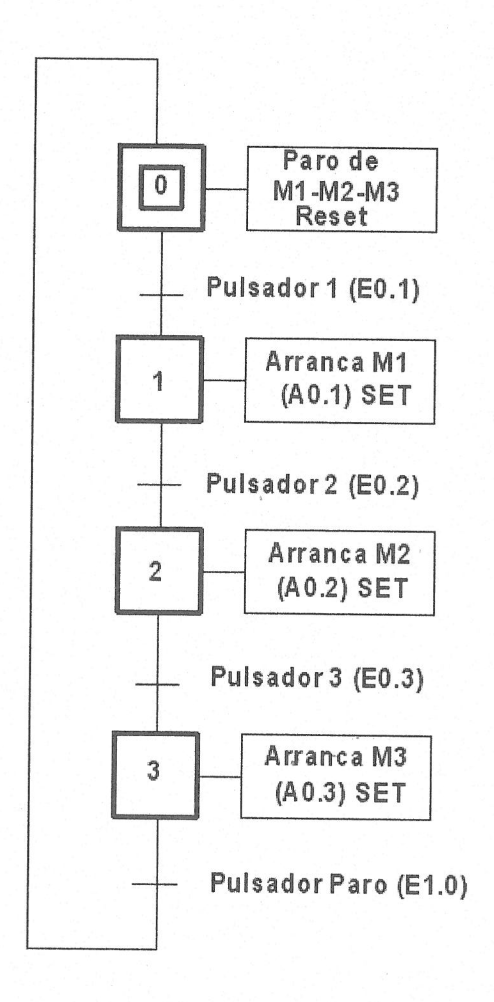


Figura 128

Se hará el programa sobre el primer grafcet (Figura 127). Se aprecia, en cualquiera de los dos, que la principal diferencia con el Ejercicio 1 es que el arranque de cada motor se realiza mediante la puesta de su salida a SET y el paro, lógicamente, a RESET. Es la única manera de poder mantener una acción después de desactivar su etapa.

TRANSICIONES

Transición Pulsador 1

```
U E0.1
U M0.0
R M0.0
S M0.1
```

Estando en la epata 0 y cumpliéndose la activación de E0.1, se resetea la etapa anterior M0.0 y se pasa a la siguiente, M0.1.

Transición Pulsador 2

```
U E0.2
U M0.1
R M0.1
S M0.2
```

Estando en la epata 1 y cumpliéndose la activación de E0.2, se resetea la etapa anterior M0.1 y se pasa a la siguiente, M0.2.

Transición Pulsador 3

```
U E0.3
U M0.2
R M0.2
S M0.3
```

Estando en la epata 2 y cumpliéndose la activación de E0.3, se resetea la etapa anterior M0.2 y se pasa a la siguiente, M0.3.

Transición Pulsador Paro

```
U E1.0
U M0.3
R M0.3
S M0.4
```

Estando en la epata 3 y cumpliéndose la activación de E1.1, se resetea la etapa anterior M0.3 y se pasa a la siguiente, M0.4.

Transición TODOS LOS MOTORES PARADOS

```
UN A0.1
UN A0.2
UN A0.3
R M0.4
S M0.0
```

Estando en la epata 4 y cumpliéndose el paro de todos los motores, se resetea la etapa anterior M0.4 y se vuelve al principio, M0.0.

ACCIONES

ETAPA 1

```
U M0.1
S A0.1
```

Arranca MOTOR1

ETAPA 2

```
U M0.2
S A0.2
```

Arranca MOTOR2

ETAPA 3

```
U M0.3
S A0.3
```

Arranca MOTOR3

ETAPA 4

```
U M0.4
R A0.1
R A0.2
R A0.3
```

Se paran todos los motores.

Ejercicio 3

Siguiendo con estos ejercicios sencillos de introducción a la programación con Grafcet, se realizará un último ejercicio en el que se va a introducir una nueva variante.

Se trata de arrancar un motor, Motor 1, al activar un pulsador (P1). A los 10 s de haber arrancado M1, arranca otro motor, Motor 2. Después de 5 s de arrancar el Motor 2, se para el Motor 1. Una vez parado el Motor 1, se puede parar el Motor 2 con un pulsador (P2). Cuando el Motor 1 y el Motor 2 se paran, arranca el Motor 3 después de 10 s de estar parados el Motor 1 y el Motor 2. Ahora el Motor 3 se puede parar con otro pulsador (P3). Una lámpara encendida indicará que no hay ningún motor conectado. El grafcet es el que se muestra en la Figura 129.

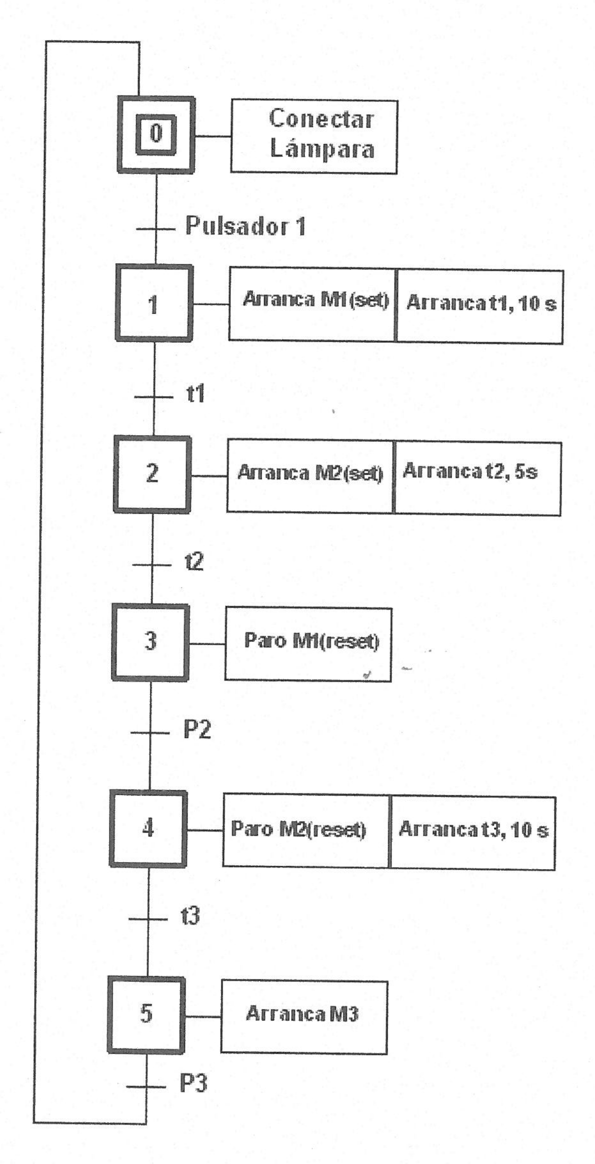


Figura 129

Se observa que tanto el Motor 1 como el Motor 2 arrancan con set y se paran con reset, debido a que ambos están conectados más de una etapa. No ocurre lo mismo con el Motor 3, que se debe parar en la siguiente etapa para ponerse en marcha.

Organización de entradas y salidas utilizadas:

ENTRADAS	SALIDAS	OTROS
P1 ..... E0.1	MOTOR 1.... A0.1	T1. Temporizador 1: 10 s
P2 ..... E0.2	MOTOR 2.... A0.2	T2. Temporizador 2: 5 s
P3 ..... E0.3	MOTOR 3.... A0.3	T3. Temporizador 3: 10 s
	Lámpara... A0.4	

PROGRAMA

Para realizar el programa hay que considerar que los temporizadores se pondrán en marcha en el modo adecuado en las acciones. El temporizador funcionará en modo SE, es decir, que una vez arrancado tardará el tiempo deseado, de 10 o 5 s, en activar su bit correspondiente. El bit del temporizador se utilizará como transición. El programa se crea con los símbolos.

TRANSICIONES

```
U P1
U M0.0
R M0.0
S M0.1

U T1
U M0.1
R M0.1
S M0.2

U T2
U M0.2
R M0.2
S M0.3

U P2
U M0.3
R M0.3
S M0.4

U T3
U M0.4
R M0.4
S M0.5

U P3
U M0.5
```

```
R M0.5
S M0.0

ACCIONES
U M0.0
= Lampara

U M0.1
L S5T#10s
SE T1
S Motor 1

U M0.2
L S5T#5s
SE T2
S Motor 2

U M0.3
R Motor 1

U M0.4
L S5T#10s
SE T3
R Motor 2

U M0.5
= Motor 3
```

DIVERGENCIAS Y CONVERGENCIAS

A la hora de hacer un grafcet, se puede organizar de diferentes maneras. A continuación se van a exponer algunos grafcet tipificados. En un grafcet se puede hacer de todo, siempre y cuando se cumplan las condiciones establecidas al principio. Estos tipos concretos de grafcet son:

Divergencia Y - Divergencia O - Convergencia Y – Convergencia O

DIVERGENCIA Y – DIVERGENCIA O

Divergencia Y

Este es un posible caso en el que de una rama se pasa a dos ramas. Desde la etapa 1 se pasa simultáneamente a dos etapas: la etapa 2 y la etapa 3. Siguiendo con el comportamiento de un grafcet, para que se pueda pasar de la etapa 1 a la etapa 2 y la etapa 3 a la vez, es necesario estar en la etapa 1 y que se cumpla la transición T1. Entonces se desactivará la etapa 1 y se activarán las etapas 2 y 3, simultáneamente.

Este es un ejemplo en el que sí estarán dos etapas activadas simultáneamente, una vez que se entre por la línea de la etapa 2 y por la línea de la etapa 3. Pero son dos líneas paralelas y no una secuencia lineal.

Para indicar que hay simultaneidad se colocan dos líneas paralelas, como se aprecia en la Figura 130.

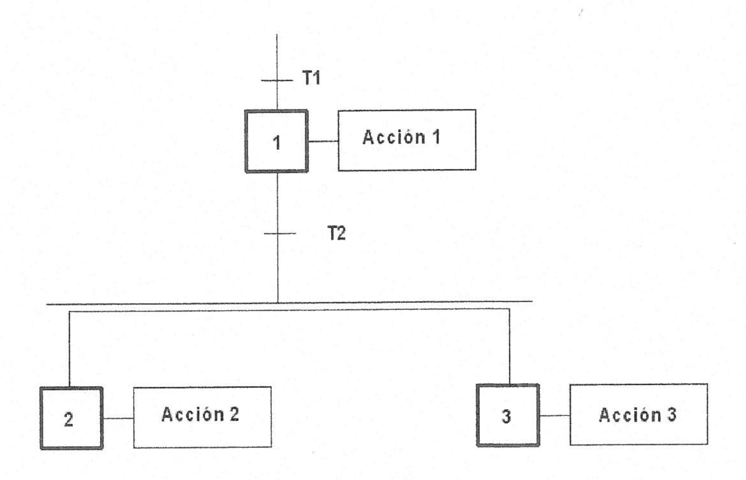


Figura 130

Se va a realizar el programa para las transiciones en el momento de la divergencia, paso de la etapa 1 a la 2 y la 3. Las marcas de las etapas 1, 2 y 3 serán M0.1, M0.2 y M0.3, respectivamente, y esto para todos los ejemplos que siguen.

```
U T2
U M0.1
R M0.1
S M0.2
S M0.3
```

Divergencia O:

En este caso se pasará de la etapa 1 a la etapa 2 o a la etapa 3. Eso dependerá de la transición que se cumpla antes. Una vez en una de las líneas (línea de la etapa 2 o línea de la etapa 3), no se podrá acceder a la otra hasta no volver al principio y pasar de nuevo por la etapa 1.

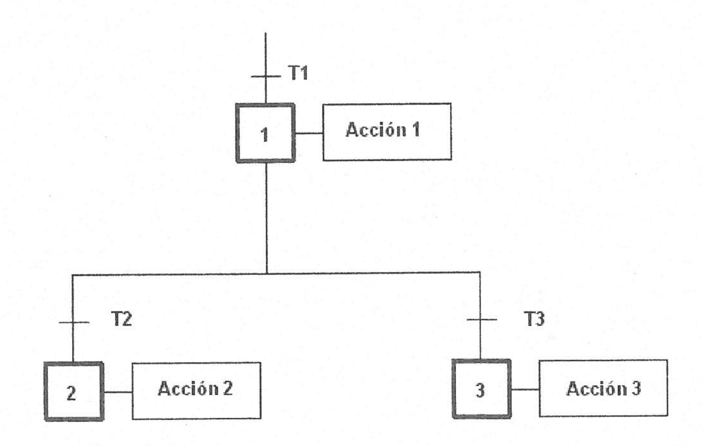


Figura 131

El programa de las transiciones es:

```
U M0.1        Esto si se entra por la línea de la etapa 2
U T2
R M0.1
S M0.2

U M0.1        Esto si se entra por la línea de la etapa 3
U T3
R M0.1
S M0.3
```

CONVERGENCIA Y - CONVERGENCIA O

Convergencia Y

Aquí se pasan de dos líneas a una. Para pasar a la etapa 4 es necesario que se esté en las dos etapas anteriores, etapa 2 y etapa 3, y además se cumpla la transición T1. Se puede decir que la primera etapa que se activa tiene que esperar a la otra (además de cumplirse la transición) para poder evolucionar a la etapa 4.

Las marcas para las etapas 2, 3 y 4 son ahora M0.2, M0.3 y M0.4, respectivamente.

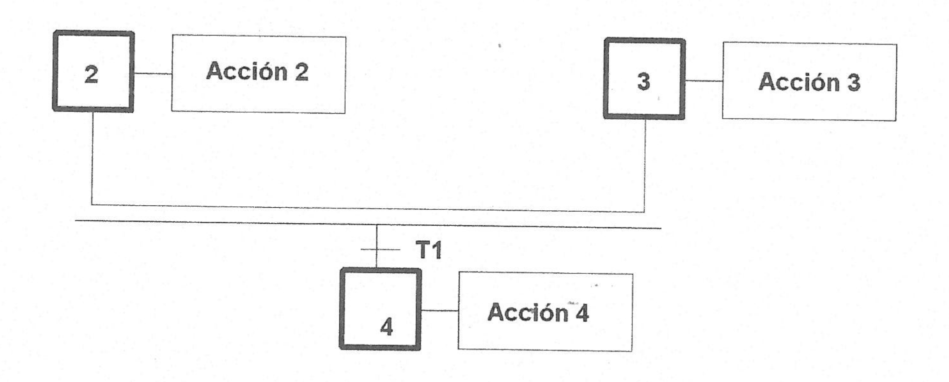


Figura 132

El programa de las transiciones es el siguiente:

```
U T1
U M0.2
U M0.3
R M0.2
R M0.3
S M0.4
```

Convergencia O

Ahora el acceso a la etapa 4 será o por la línea de la etapa 2 o por la línea de la etapa 3. Una vez activada la etapa 2 o la etapa 3, y cumplida la transición correspondiente, T1 o T2 respectivamente, el grafcet continuará por la etapa 4.

Las instrucciones de las transiciones son:

```
U T1
U M0.2
R M0.2
R M0.3
S M0.4

U T2
U M0.3
R M0.2
R M0.3
S M0.4
```

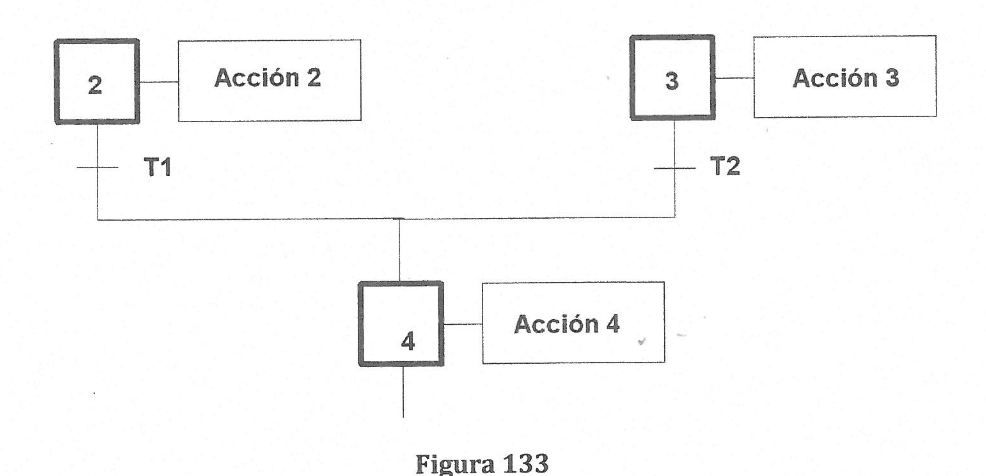


Figura 133

Otras posibilidades. Saltos

Con un grafcet es posible llevar a cabo cualquier opción, siempre que se cumpla con el rigor que exige el principio establecido para su correcto uso.

En los siguientes ejemplos de saltos se va a realizar el programa de transiciones. Son tan solo dos ejemplos, pero se pueden presentar muchas más posibilidades. Únicamente se realizan las transiciones en las que participa el salto, la salida del salto y la llegada. Si se considera necesario, se pueden utilizar flechas para indicar y aclarar el sentido del grafcet.

Paso de la etapa 2 a la 3:

```
U T3
U M0.2
R M0.2
S M0.3
```

Paso de la etapa 5 a la 6:

U T6

U M0.5

R M0.5

S M0.6

Paso de la etapa 5 a la 51:

U T51

U M0.5

R M0.5

S M5.1

Paso de la etapa 51 a la etapa 3.

U T52

U T3

U M5.1

R M5.1

S M0.3

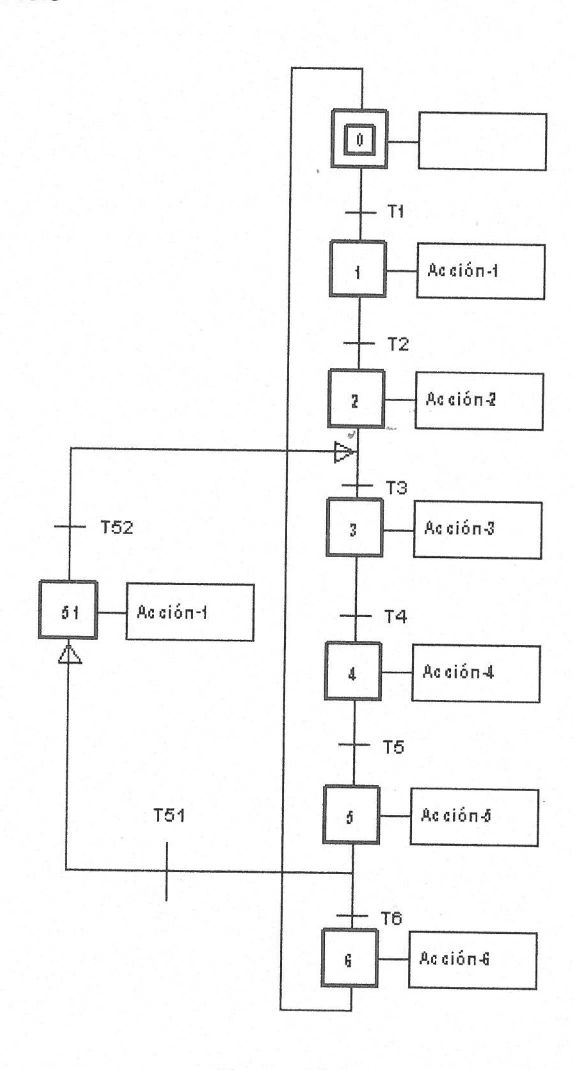


Figura 134

Se observa que, estando en la etapa 2, solo hay un camino, que es seguir por la etapa 3. En la etapa 5 hay dos caminos (Divergencia O): o se sigue por la etapa 6 o por la etapa 51. En ambos casos deberá resetearse la etapa 5.

Para pasar de la etapa 51 a la etapa 3 se debe cumplir la transición T52 y la T3. También podría darse el caso de que solo se debiera cumplir la transición T52. En este ejemplo no ocurre esta situación, según se aprecia. En cualquiera de las dos opciones, una vez activada la etapa 3, se debe desactivar la etapa 51, que es la que estaba activada.

Otro caso será con el salto hacia abajo, como es el caso de la Figura 135.

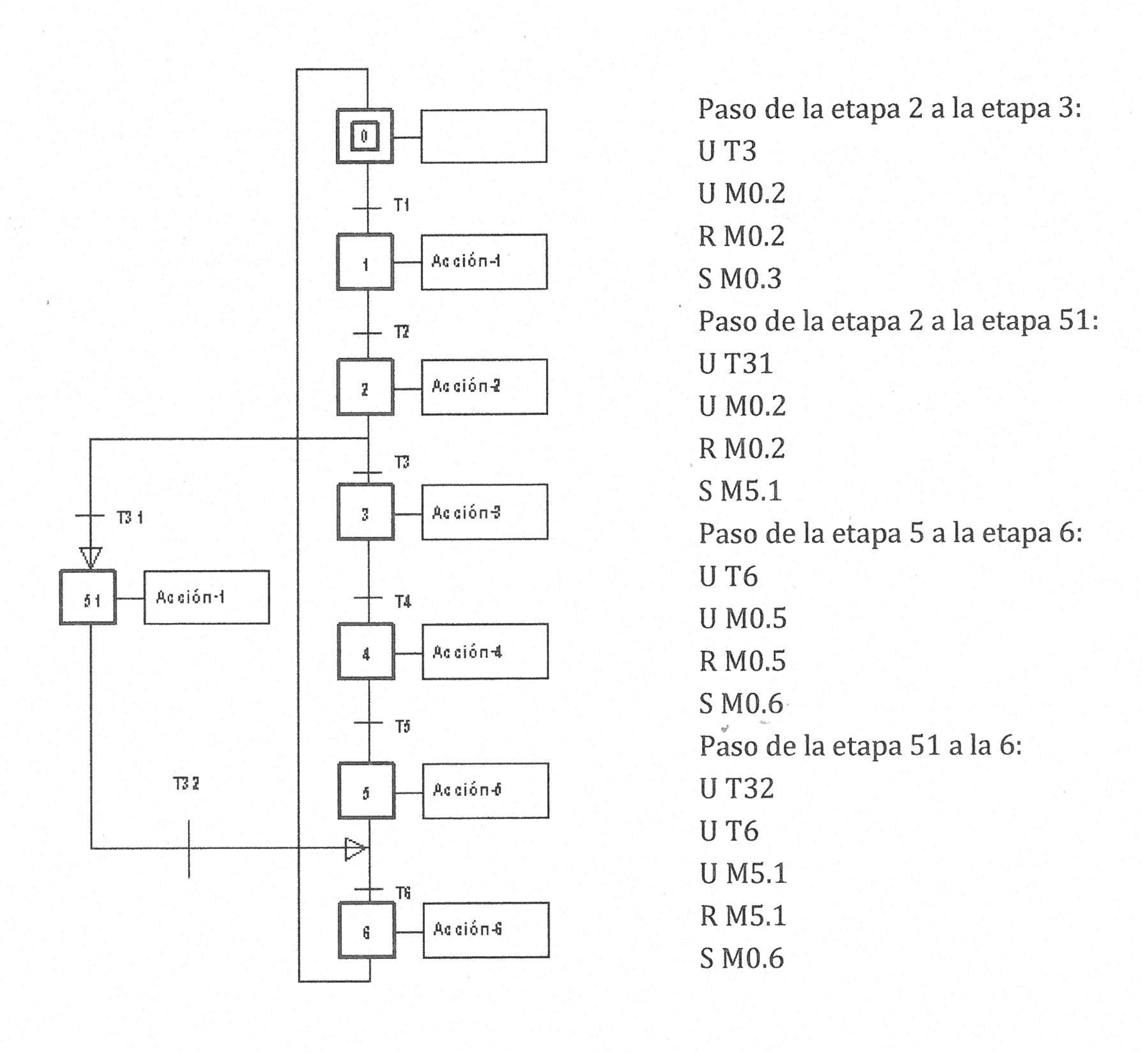


Figura 135

FUNCIONES EN GRAFCET

Si en un grafcet debe ejecutarse alguna función (FC, SFC o FB), se puede llevar a cabo como una acción. La forma gráfica de hacerlo en el grafcet se observa en la Figura 136.

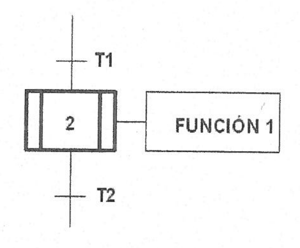


Figura 136

Cuando se deba ejecutar la etapa 2, se abandonará el grafcet general y se ejecutará la función FC 1. Cuando termine, la función volverá de nuevo a la etapa 2 y solo evolucionará cuando, estando en la etapa 2, se cumpla la transición T2.

EJERCICIOS GRAFCET

A continuación se van a realizar algunos ejercicios para practicar con GRAFCET. Igualmente se propondrán otros. Puede haber otras soluciones diferentes a las planteadas aquí y que sean igualmente válidas. Lo importante es que la opción tomada se compruebe y funcione respondiendo al enunciado.

NO OLVIDAR CARGAR EL OB100 CON LA ACTIVACIÓN DE LA MARCA M0.0, COMO MÍNIMO

Ejercicio 1

Al accionar un pulsador, P1, arrancan tres motores M1, M2 y M3 en cascada con un tiempo de diferencia de 1 s. Al pulsar P2 y al cabo de 5 s, se para M2 y, al cabo de 15 s (de accionar P2), se para M3.

Al cabo de 7 s de estar los dos parados (M2 y M3) se para M1.

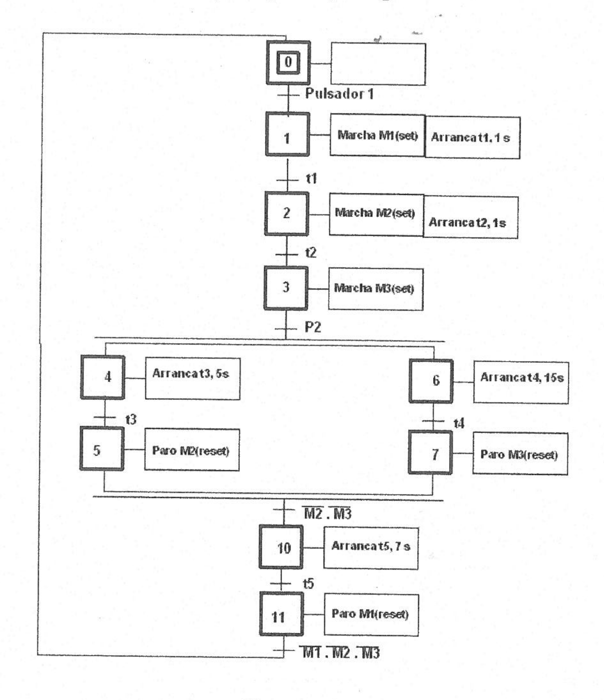


Figura 137

Distribución de E/S

ENTRADAS	SALIDAS	OTROS
Pulsador 1.... E0.1	Motor 1.... A0.1	Temporizador T1, 1 s
Pulsador 2.... E0.2	Motor 2.... A0.2	Temporizador T2, 1 s
	Motor 3....A0.3	Temporizador T3, 5 s
		Temporizador T4, 15 s
		Temporizador T5, 7 s

```
//TRANSICIONES
U   Pulsador 1
U   M0.0
R   M0.0
S   M0.1

U   T1
U   M0.1
R   M0.1
S   M0.2

U   T2
U   M0.2
R   M0.2
S   M0.3

U   Pulsador 2
U   M0.3
R   M0.3
S   M0.4
S   M0.6

U   T3
U   M0.4
R   M0.4
S   M0.5

U   T4
U   M0.6
R   M0.6
S   M0.7

UN  A0.2
UN  A0.3
U   M0.7
U   M0.5
R   M0.5
```

```
//ACCIONES
U   M0.1
L   S5T#1S
S   Motor 1
SE  T1

U   M0.2
L   S5T#1S
S   Motor 2
SE  T2

U   M0.3
S   Motor 3

U   M0.4
L   S5T#5S
SE  T   3

U   M0.5
R   Motor 2

U   M0.6
L   S5T#15S
SE  T4

U   M0.7
R   Motor 3

U   M1.0
L   ST#7S
SE  T5

U   M1.1
R   Motor 1
```

```
R   M0.7
S   M1.0

U   T5
U   M1.0
R   M1.0
S   M1.1

UN  Motor 1
UN  Motor 2
UN  Motor 3
U   M1.1
R   M 1.1
S   M0.0
```

Ejercicio 2

Se pretende realizar el control de dos carretillas que arrancan activando el pulsador P. Ambas permanecen a la derecha hasta pulsar P. En ese momento los dos móviles se desplazan a la izquierda. El primer móvil que llega a la izquierda espera al otro. Estando los dos a la izquierda, esperan 5 s hasta que de nuevo hacia la derecha. Cuando cada uno llega a la derecha, se paran 1 s e invierten su sentido de giro sin esperar al otro. El proceso sigue hasta que se pulsa STOP.

Cuando se pulsa STOP, ambos móviles retornan a la derecha y se termina el ciclo. Si al accionar STOP alguna carretilla está a la izquierda, se esperarán 5 s a arrancar.

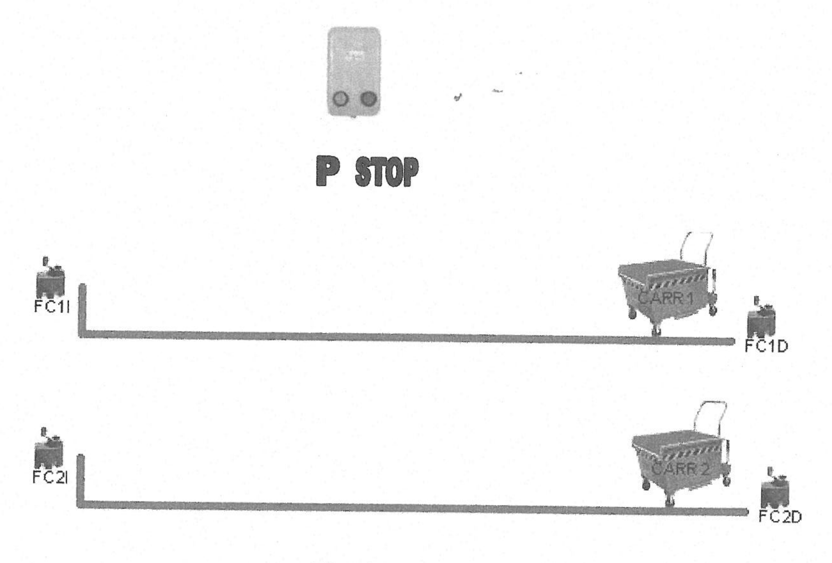


Figura 138

ENTRADAS	SALIDAS	OTROS
Pulsador P.... E0.0	Motor carretilla 1 izda..... A0.0	Temporizador T1... 5 s
FC1I.... E0.1	Motor carretilla 2 izda..... A0.1	Temporizador T2... 1 s
FC2I.... E0.2	Motor carretilla 1 dcha..... A1.0	Temporizador T3... 1 s
FC1D.... E1.1	Motor carretilla 2 dcha..... A1.1	
FC2D.... E1.2		
STOP.... E1.7		

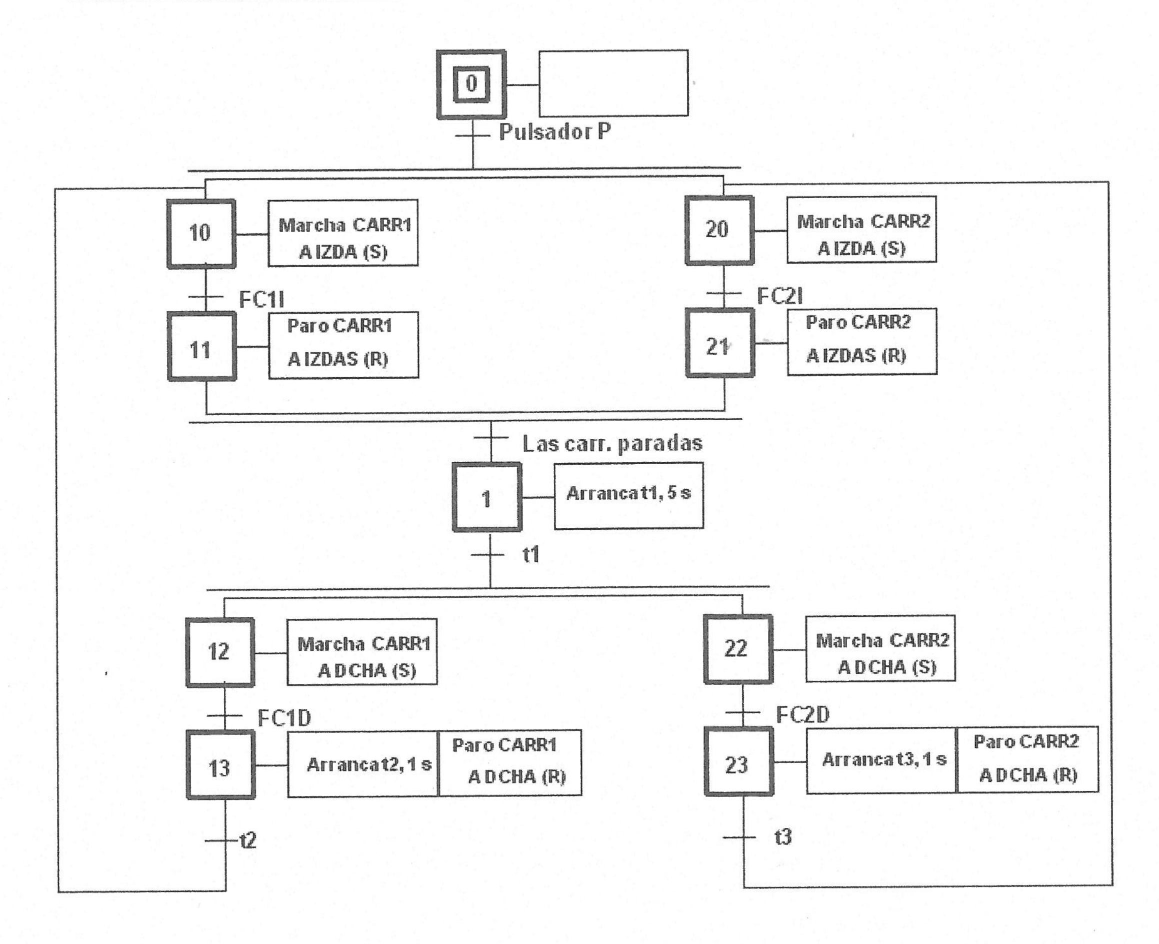


Figura 139

En el grafcet no aparece la circunstancia del STOP, por lo que en ese caso las dos carretillas deben volver a la derecha, estén donde estén. Esta situación se pondrá en el programa independiente al grafcet.

```
// TRANSICIONES
U Pulsador P
U M0.0
R M0.0
S M1.0
S M2.0

U FC1I
U M1.0
R M1.0
S M1.1

U FC2I
U M2.0
R M2.0
S M2.1

UN Motor carretilla 1 Izda
UN Motor carretilla 2 Izda
U M1.1
U M2.1
R M1.1
R M2.1
S M0.1

U T1
U M0.1
R M0.1
S M1.2
S M2.2

U FC1D
U M1.2
R M1.2
S M1.3

U FC2D
U M2.2
R M2.2
S M2.3

U T2
U M1.3
R M1.3
S M1.0

U T3
U M2.3
R M2.3
S M2.0
```

```
//ACCIONES
U M1.0
S Motor carretilla 1 Izda

U M1.1
R Motor carretilla 1 Izda

U M2.0
S Motor carretilla 2 Izda

U M2.0
R Motor carretilla 2 Izda

U M0.1
L S5T#5S
SE T1

U M1.2
S Motor carretilla 1 Dcha

U M2.2
S Motor carretilla 2 Dcha

U M1.3
L S5T#5S
SE T2
R Motor carretilla 1 Dcha

U M2.3
L S5T#1S
SE T3
R Motor carretilla 2 Dcha
```

En el programa anterior no se ha tenido en cuenta el pulsador de STOP. Se tendrá en cuenta ahora de forma independiente al grafcet realizado.

Cuando se pulsa STOP, se deben parar las acciones del grafcet y llevar las carretillas a la derecha si se encuentran en marcha. Si están paradas a la derecha, no hará nada y si están a la izquierda, entrará un temporizador de 5 s para que avancen a la derecha.

Cualquiera de estas situaciones debe concluir llevando el sistema a la etapa 0 del grafcet.

```
U STOP          //Se hace del pulsador STOP un interruptor.
R M0.0
R M0.1
R M1.0
R M1.1
R M2.0
R M2.1
R M1.2
R M1.3
R M2.2
R M2.3
R Motor carretilla 1 Dcha
R Motor carretilla 2 Dcha
R Motor carretilla 1 Izda
R Motor carretilla 2 Izda
S M40.0

UN FC1I         // Si se pulsa STOP y la carretilla 1 no está a la izda., se desplazan a la dcha.
U M40.0
S Motor carretilla 1 Dcha
R Motor carretilla 1 Izda

UN FC2I         //Si se pulsa STOP y la carretilla 2 no está a la izda., se desplazan a la dcha.
U M40.0
S Motor carretilla 2 Dcha
R Motor carretilla 2 Izda

U FC1I          //Si se pulsa STOP y la carretilla 1 está a la izda., al cabo de 5 s se mueve a la
U M40.0         //derecha.
L S5T#5s
SE T4

U T4
S Motor carretilla 1 Dcha

U FC2I          //Si se pulsa STOP y la carretilla 2 está a la izda., al cabo de 5 s se mueve a la
U M40.0         //derecha
L S5T#5s
SE T5

U T5
S Motor carretilla 2 Dcha
```

```
U FC1D                 // Para carretilla 1 a la derecha
R Motor carretilla 1 Dcha

U FC2D                 //Para carretilla 2 a la derecha
R Motor carretilla 2 Dcha

U M40.0      //Si se pulsa STOP y están las dos carretillas a la derecha, se activa la etapa 0 y
U FC1D       //se resetea la M40.0 que hace de STOP un interruptor
U FC2D
S M0.0
R M40.0
```

Ejercicio 3

En el ejercicio anterior se requiere añadir un paro (PARO), de forma que al activarlo se pare el proceso y al pulsar marcha (P) siga realizando la misma maniobra que tenía antes del paro. El pulsador de PARO es otro distinto al de STOP y el de marcha será el mismo pulsador P del ejercicio anterior.

Para resolver este nuevo planteamiento, no será necesario modificar el grafcet del ejercicio anterior. Se hará de forma que cuando se pulse PARO, se ejecute una función (Función de paro) y cuando se pulse P, después de haber pulsado PARO, se entrará en otra función (Función de marcha).

Hay que cuidar que cuando se pulse P sin haber pulsado antes PARO, no debe entrar en la función de marcha. Tampoco se debe entrar en la función de marcha si no se ha activado antes el PARO.

Con esas premisas se va a realizar el programa. Al activar el PARO, lo que se debe hacer, y en este orden, es:

1) Guardar en memoria (marcas) los valores de las etapas actuales (las marcas de cada etapa).
2) Guardar el estado de las salidas.
3) Poner a cero (borrar) las marcas y las salidas.

Al activar marcha (P) lo que se debe hacer, y en este orden, es:

1) Volver a sacar los valores de las marcas de cada etapa.
2) Volver cada salida a su valor de antes de pulsar PARO.

FUNCIÓN PARO (FC1):

SET S M50.0	Activa la marca M50.0 para tener constancia de haber entrado en la FC y pulsado PARO *(huella)*.
L MD0 T MD100	Guarda todas las marcas del grafcet en otras marcas.

L 0 T MD0	Borra el estadio del grafcet poniendo todas las marcas a cero.
AW0 T MW 105	Guarda todas las salidas en marcas.
L 0 T AW0	Borra las salidas para parar.

FUNCIÓN DE MARCHA (FC2):

SET R M50.0	Reinicia la huella.
L MD100 T MD0	Restituye los valores del grafcet.
L MW105 T AW0	Restituye los valores de las salidas.

AÑADIR EN EL OB1:

U PARO UN M50.0 CC FC 1	Si se pulsa PARO, ejecuta la función de paro. Con la marca cerrada M50.0 se evita que se pueda pulsar dos veces el paro. Si no se pone y se pulsa más de una vez, guardaría 0 para marcas y salidas.
U P1 U M50.0 CC FC2	Si se pulsa P1 (marcha) y antes se ha activado PARO, se ejecuta FC2

Ejercicio 4

Se desea activar y desactivar una carga con un único pulsador. Es decir, que si la carga está activada, al accionar P se desactiva, y, si está desactivada, al pulsar P, se activa.

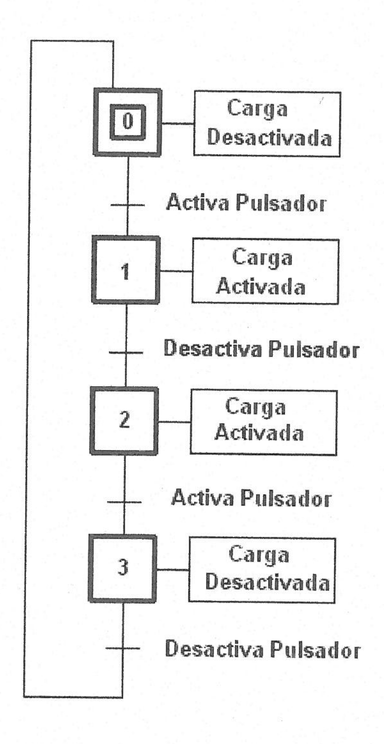


Figura 140

NO OLVIDAR CARGAR EL OB100 CON LA ACTIVACIÓN DE LA MARCA M0.0

Como pulsador se utiliza la entrada E0.0 y como carga la salida A0.0.

```
U E0.0
U M0.0
R M0.0
S M0.1

UN E0.0
U M0.1
R M0.1
S M0.2

U E0.0
U M0.2
R M0.2
S M0.3

UN E0.0
U M0.3
R M0.3
S M0.0

U M0.1
O M0.2
= A0.0
```

Ejercicio 5

Una puerta se abre al pisar una alfombra. Una vez abierta, se mantiene en esa situación 8 s, tras lo cual se cierra automáticamente. Si en el transcurso del tiempo se cierra a alguien que atraviesa la puerta, se vuelve a abrir gracias a una fotocélula que detecta a la persona. De esta forma vuelve a repetirse el ciclo.

ENTRADAS	SALIDAS	OTROS
Alfombra.... E0.0	Motor puerta abrir... A0.0	Temporizador T1... 8s
FC Puerta cerrada... E0.1	Motor puerta cerrar... A0.1	
FC Puerta abierta ... E0.2		
Fotocélula... E0.3		

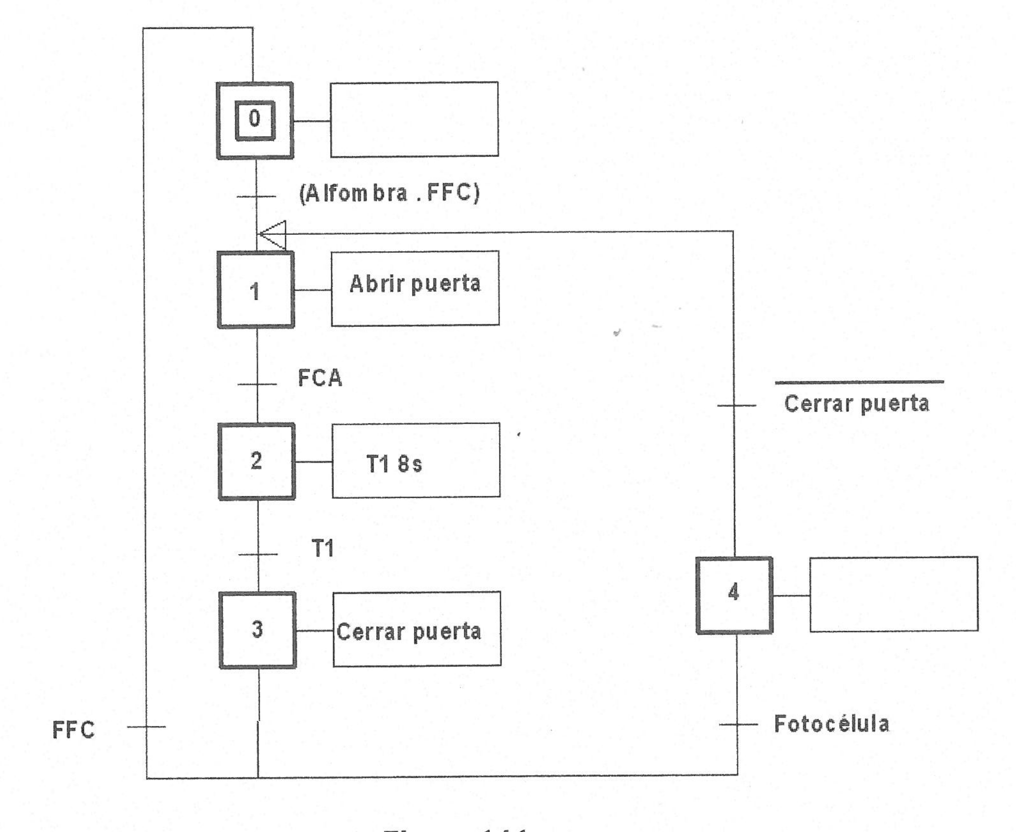


Figura 141

TRANSICIONES

```
U   E 0.0
U   E 0.1
U   E0.3
U   M0.3
R   M0.3
S   M0.4

U   M0.0
R   M0.0
S   M0.1

U   E0.2
U   M0.1
R   M0.1
S   M0.2

U   T1
U   M0.2
R   M0.2
S   M0.3
U   E0.1

U   M0.3
R   M0.3
S   M0.0

UN  A0.1
U   M0.4
R   M0.4
S   M0.1
```

ACCIONES

```
U   M   0.1
=   A   0.0

U   M   0.3
=   A   0.1

U   M   0.2
L   S5T#8S
SE  T   1
```

Ejercicios PROPUESTOS

1.º GRÚA DE DOS CICLOS

Se trata de automatizar el funcionamiento de una grúa de forma que desde la posición inicial de reposo, y accionando el pulsador PM, la grúa sube el gancho estando en la posición izquierda. Seguidamente se desplaza hacia la derecha y el gancho baja. Aquí termina el primer ciclo

Estando el gancho en la posición inferior y la grúa a la derecha se activa un tiempo de espera de 5 s. Terminado ese tiempo, sube el gancho y la grúa retorna a la posición izquierda, baja de nuevo el gancho concluyendo el segundo ciclo y volviendo a la posición de reposo.

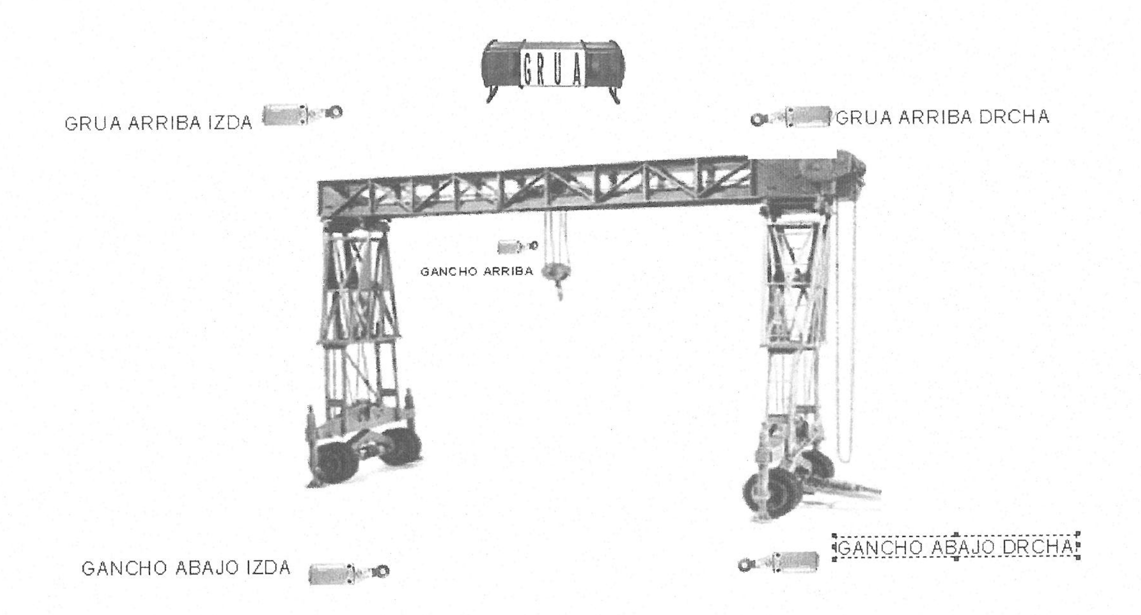


Figura 142

2.º PORTÓN CORREDIZO

Se desea controlar con un pulsador, P, la apertura y el cierre de un portón corredizo. Cuando se acciona el pulsador P, y el portón está cerrado, se debe abrir. Y si está abierto, al accionar P, se cierra. Si se está abriendo, al pulsar P, se para y solo podrá seguir abriéndose si se pulsa de nuevo P. Igualmente, si se está cerrando, al pulsar P se parará y solo podrá seguir cerrándose si se vuelve a accionar P.

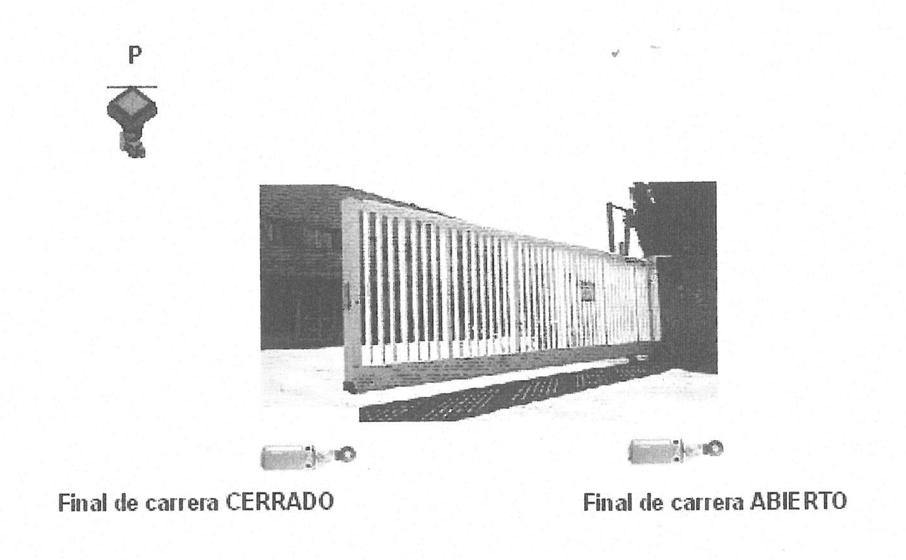


Figura 143

3.º SEMÁFORO CON PULSADOR PARA PEATONES



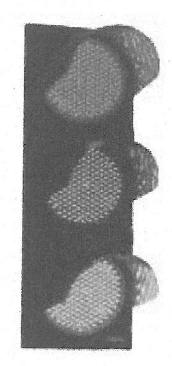




Figura 144

Se desea automatizar un semáforo para peatones. El semáforo para los coches está normalmente verde hasta que un peatón desea cruzar y acciona el pulsador P. En este momento se desencadena la siguiente secuencia:

- Al cabo de 10 s de pulsar P, el semáforo de coches pasa a ámbar.
- Se mantiene en ámbar durante 3 s, después de lo cual se enciende la luz roja durante 10 s.
- A la vez se enciende el verde para peatones.
- Está en verde para peatones durante 8 s fijo y 2 s más intermitente a una frecuencia de 0,5 s.
- El semáforo vuelve a la posición de normal, es decir, luz verde para coches y roja para peatones.

13. ALARMAS

INTRODUCCIÓN

Es importante poder gestionar de una forma asíncrona la ejecución del PLC, no solo por la propia ejecución del programa, sino de una forma independiente a él. Es decir, hacer que se interrumpa por motivos externos.

De eso va a tratar este tema. Se estudiarán las alarmas horarias de retardo y cíclicas. Todas estas alarmas utilizan un programa que se escribe en un OB concreto. Este OB se ejecuta cuando se requiere, en función de lo que dicha alarma hace y tal como se ha configurado. Son subrutinas o funciones que se ejecutan, no por el OB1, sino mediante una interrupción.

Todas las alarmas se pueden simular.

Lo que aquí se va a ver es para el STEP7 V5.5 y TIA PORTA para el PLC S7 300.

ALARMAS HORARIAS

Las alarmas horarias son subprogramas que se ejecutan en función de cómo se han configurado. Puede ser que se ejecuten un día a una hora determinada, cada cierto tiempo o una sola vez.

Sirve para recordatorio de eventos o para activar cargas cada cierto tiempo, o en un momento determinado; también de forma cíclica, para mantenimientos preventivos por ejemplo.

Como alarmas horarias se disponen de ocho OB. En la serie 300 solo uno, el OB 10.

Estas alarmas horarias se pueden utilizar de dos formas diferentes:

- A nivel configuración de *hardware* en Step 7.
- A nivel programa con el uso de la SFC 30.

El uso mediante la configuración de las alarmas desde el *hardware* de Step 7 es más rápido y cómodo, pero por el contrario es menos práctico. A nivel de programa puede ser modificado/actualizado desde fuera del PLC, mediante una pantalla gráfica, por ejemplo.

Los intervalos de uso de los OB en ambos caso son:

Una vez / Cada minuto / Cada hora / Cada día / Cada semana / Cada mes / Cada año

CONFIGURACIÓN DE OB10 DESDE EL *HARDWARE* DE STEP 7 V5.5

Para poder utilizar el OB10 mediante la configuración del *hardware* de Step 7 se debe ir a la ventana de *hardware* y, haciendo clic dos veces sobre la CPU, acceder a las propiedades.

En la Figura 145 se muestra la ventana **Propiedades**.

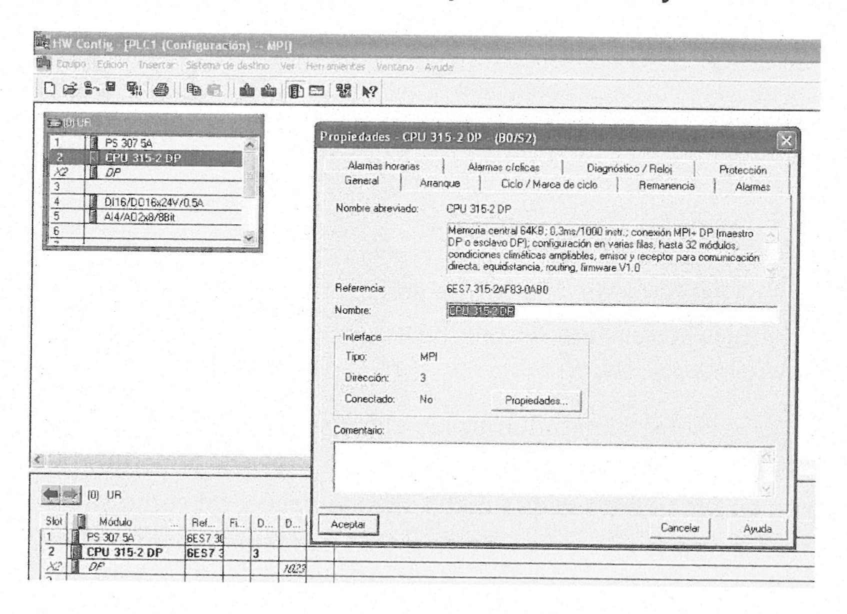


Figura 145

Se accede a la pestaña **Alarmas horarias** y allí se realiza la configuración. Una vez hecha, se compila y se envía al PLC.

Propiedades - CPU 315-2 DP - (B0/S2)

General | Arranque | Ciclo / Marca de ciclo | Remanencia | Alarmas
Alarmas horarias | Alarmas cíclicas | Diagnóstico / Reloj | Protección

	Prioridad	Activa	Periodicidad	Fecha de arr.	Hora	Imagen parcial del proceso
OB10:	2	☑	Una vez	3/05/2012	13:15	---
OB11:	2	☐	Ninguna	01/01/199	00:00	---
OB12:	2	☐	Ninguna	01/01/199	00:00	---
OB13:	2	☐	Ninguna	01/01/199	00:00	---
OB14:	2	☐	Ninguna	01/01/199	00:00	---
OB15:	2	☐	Ninguna	01/01/199	00:00	---
OB16:	2	☐	Ninguna	01/01/199	00:00	---
OB17:	2	☐	Ninguna	01/01/199	00:00	---

Aceptar | Cancelar | Ayuda

Figura 146

Se debe marcar la casilla **Activa** para que se pueda utilizar la alarma horaria (OB10). La prioridad no se puede cambiar. Se indica la periodicidad deseada, el día y la fecha. Si se selecciona alguna opción cíclica (cualquier otra que sea diferente a la opción **Una vez**), la hora y el día determinan el momento de inicio del ciclo, es decir, la primera vez que se ejecutará.

Para poder utilizar la alarma horaria del OB10 es evidente que se debe escribir en él el programa que se desee ejecutar cuando se cumpla la hora y el día, y posteriormente cargarlo en el PLC. Si no está el OB10 y se ha activado el PLC, se irá a STOP. El OB10 se puede tomar de

la librería estándar (en **Bloques de organización**). También se puede crear en bloque como un bloque de organización más, indicando 10.

Para llevar a cabo la prueba, se va a realizar el siguiente ejercicio.

En el OB10 se va a poner a uno una salida y en OB1 se borrará con un pulsador. Se puede seleccionar el día y la hora, y ejecutarlo de forma cíclica cada minuto.

OB1:	OB10:
U E0.0	SET
R A0.0	S A0.0

CONFIGURACIÓN DE OB10 DESDE EL *HARDWARE* EN TIA PORTAL

En TIA PORTAL el procedimiento es igual, pero cambia el lugar en el que debe realizarse. El OB10 se selecciona desde el árbol del proyecto, en **Bloques de programa/Agregar nuevo bloque**. En la ventana que aparece se selecciona OB y se muestran todos los OB disponibles, además del OB1. Se busca el OB10 que aparece en la carpeta **Time of day.** (Figura 147).

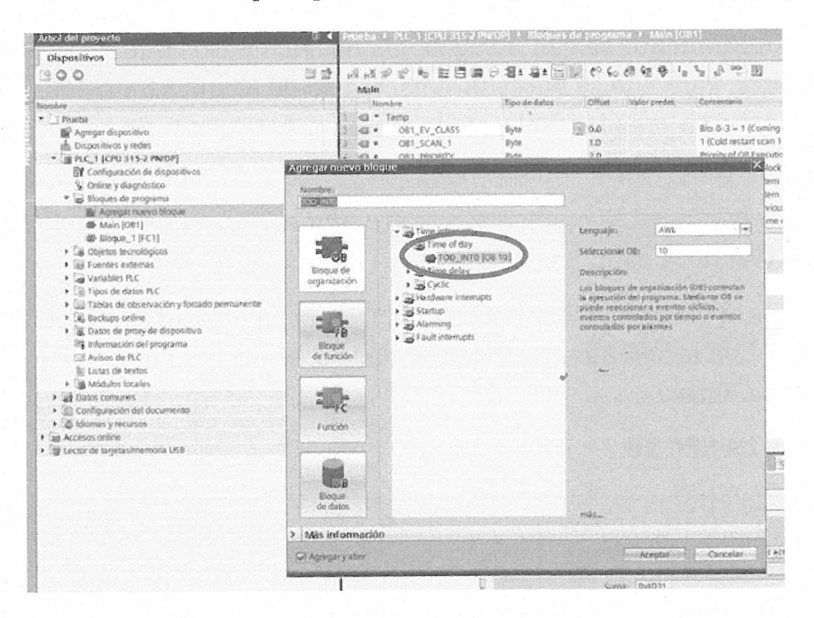


Figura 147

Ahora, y desde las propiedades del PLC, se debe configurar la alarma OB10. Se selecciona **Alarmas horarias** y se activa la alarma OB10. En **Ejecución** se indica la periodicidad. También se debe indicar la hora de arranque, de la misma forma que se hacía en STEP7 V5.5. En la Figura 148 se muestra un ejemplo.

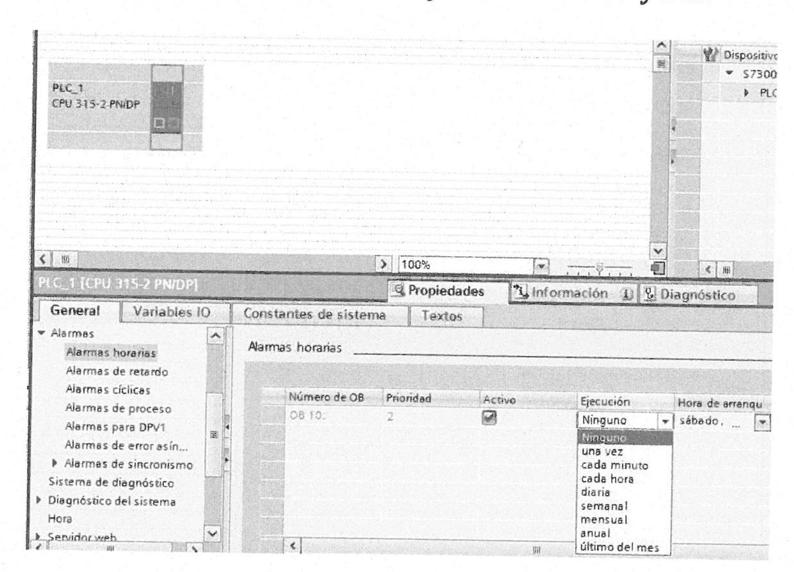


Figura 148

Se puede realizar el mismo programa que se ha visto para STEP7 V5.5.

CONFIGURACIÓN DE OB 10 POR *SOFTWARE* CON STEP7 V5.5

Mediante este procedimiento se van a seguir los mismos pasos que en el apartado anterior, pero a nivel *software*. Para ello se necesitará hacer uso de algunas funciones de sistema (SFC). Estas funciones están ya creadas, solo se deben parametrizar y no se cargan sobre el PLC, porque están dentro del autómata.

Para poder realizar la ejecución mediante este sistema es importante que NO ESTÉ MARCADA LA PESTAÑA **ACTIVA** de la pantalla de *hardware* (Figura 146).

Se deberá utilizar la SFC 28 y la SFC 30. Con la SFC 28 se indicarán la periodicidad, y la fecha y hora de activación. Con la SFC 30 se activará la alarma, es decir, lo que se hacía en el *hardware* al marcar la pestaña **Activa**.

PARÁMETROS DE LA SFC 28

PARÁMETRO	TIPO DE DATO	SIGNIFICADO
OB_NR	INT	Número del OB de alarma que se desea utilizar (OB10 a OB17). EN S7 300 solo el OB 10.
SDT	DT (DATE_AN_TIME)	Fecha y hora de arranque. De la hora de arranque que se haya especificado se ignoran los segundos y los milisegundos, y se ponen a cero. Si desea definir un arranque mensual de un OB de alarma horaria, para la fecha de arranque solo se pueden indicar los días 1, 2, ... 28.

PARÁMETRO	TIPO DE DATO	SIGNIFICADO
PERIOD	WORD	Periodicidad que se desea asignar: W#16#0000 = una vez W#16#0201 = cada minuto W#16#0401 = cada hora W#16#1001 = diaria W#16#1201 = semanal W#16#1401 = mensual W#16#1801 = anual W#16#2001 = al final del mes
RET_VAL	WORD	Si ocurre un error al procesar la función, el parámetro actual de RET_VAL contiene un código de error.

En el parámetro SDT se debe incluir la fecha y la hora de ejecución. Este dato debe estar en un tipo de formato de dato compuesto llamado DATE-TIME, que se coloca en un símbolo definido por defecto en el OB1 llamado #OB1_DATE_TIME, tal como se ve en la Figura 149.

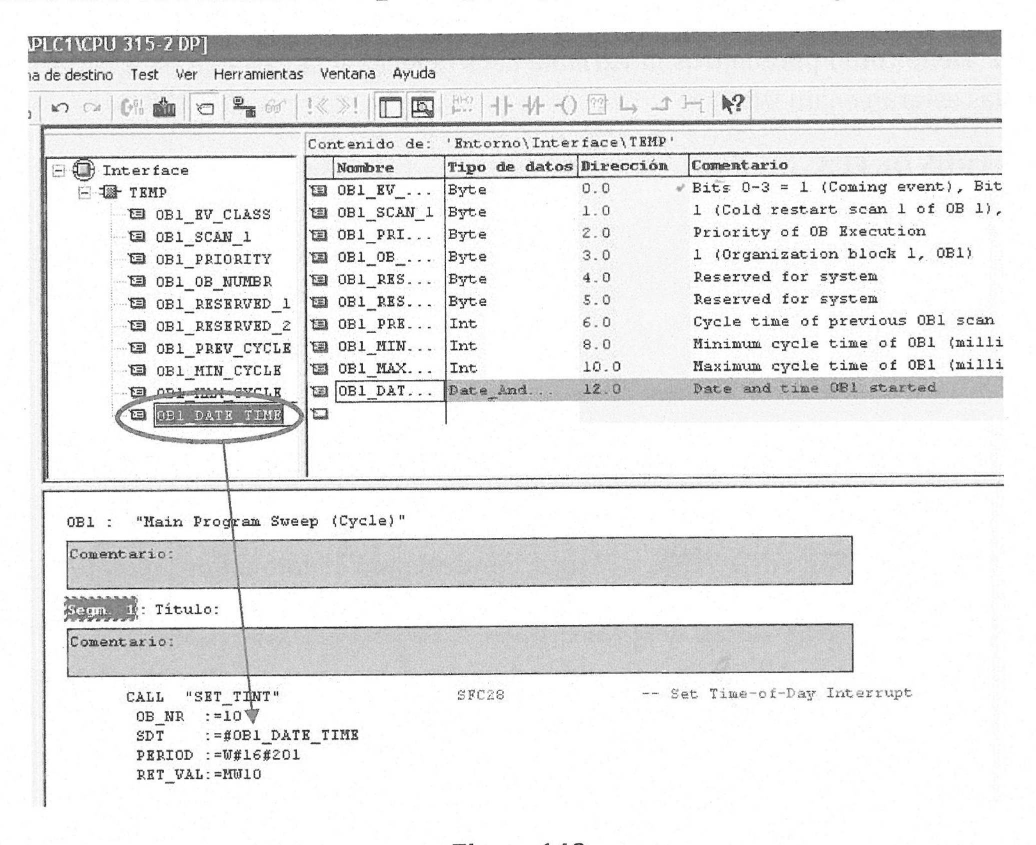


Figura 149

PARÁMETROS DE LA SFC30

PARÁMETRO	TIPO DE DATO	SIGNIFICADO
OB_NR	INT	Número del OB que se desea activar (habilitar) (OB10 a OB17). En el 300 solo el OB10.
RET_VAL	WORD	Si ocurre un error al procesar la función, el parámetro actual de RET_VAL contiene un código de error.

En el siguiente ejemplo se va a utilizar la alarma horaria en este modo.

Para poder introducir la fecha y la hora se debe usar un tipo de dato que es compuesto y se denomina DATE-TIME. Se va a utilizar un DB para introducir la hora y la fecha. Para ello se podría utilizar una pantalla gráfica y desde allí modificar tanto la hora como la fecha. Como en este libro no se tratan las pantallas gráficas, se podrán modificar esos datos desde la propia DB, sin necesidad de modificar el programa OB1.

En el DB se introducirán la hora y fecha por separado. Para unir esos dos datos simples en un solo dato compuesto del tipo DATE-TIME, que es el que necesitamos, se va a utilizar una función que realiza esa labor. La función es la FC3 y se deberá tomar de la librería de funciones: **Standard Library, IEC Function Blocks, Bloques.** Esta FC3 debe cargarse en el autómata. Tiene como parámetros la variable para la fecha y la variable para la hora. En este caso ambas estarán en un DB.

PARÁMETROS DE FC3

PARÁMETRO	TIPO DE DATO	SIGNIFICADO
IN1	DATE	Lugar donde está la fecha
IN2	TIME_OF_DAY (TOD)	Lugar donde está la hora (hora del día)
RET_VAL	DATE_AND_TIME	Lugar donde dejará el dato unido (OB1_DATE_TIME)

La SFC 28 y SFC 30 se deberán coger de la librería de funciones: **Standard Library, IEC Function Blocks, Bloques.**

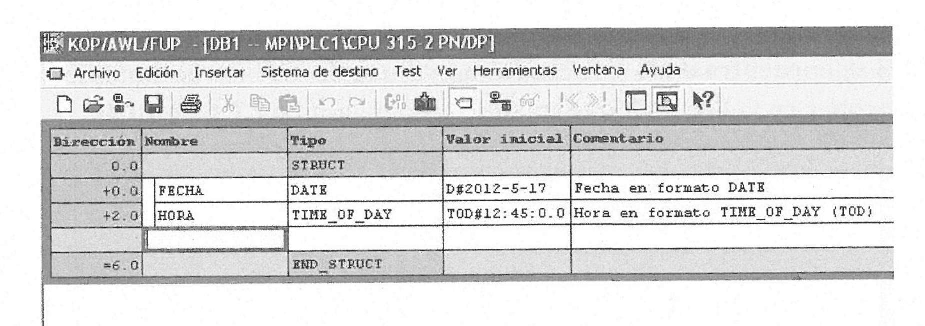
Programa OB1

Código	Comentario
CALL "D_TOD_DT"	FC3 que une los formatos simples de fecha y hora en un formato compuesto de hora y fecha.
IN1 :=DB1.DBW0	FECHA
IN2 :=DB1.DBD2	HORA
RET_VAL:=#OB1_DATE_TIME	Copiar y pegar de la zona de la interfaz del OB1(parte superior).
U E0.0 FP M 0.0 SPB l1	Con E0.0 se cambiará la hora y la fecha ejecutando la SC28.
U E0.1 FP M0.1 SPB l2	Con E0.1 se activa el uso de la alarma horaria ejecutando la SFC30.
U E1.0 R A0.0 BEA	Con E1.0 se resetea la salida que se activa con la alarma horaria cada minuto.
L1: CALL "SET_TINT" OB_NR :=10 SDT :=#OB1_DATE_TIME PERIOD :=W#16#201 RET_VAL:=MW10	SFC 28 Aquí se cambia la fecha y hora con #OB1_DATE_TIME Ese valor se carga con los valores del DB1 en la FC3. Cada minuto. Marca de palabra 10 utilizada para avisar de un posible error.
L2: CALL "ACT_TINT" OB_NR :=10 RET_VAL:=MW20	SFC 30

PROGRAMA OB 10

```
SET
S A0.0
```

DB1



Dirección	Nombre	Tipo	Valor inicial	Comentario
0.0		STRUCT		
+0.0	FECHA	DATE	D#2012-5-17	Fecha en formato DATE
+2.0	HORA	TIME_OF_DAY	TOD#12:45:0.0	Hora en formato TIME_OF_DAY (TOD)
=6.0		END_STRUCT		

Figura 150

CONFIGURACIÓN DE OB 10 POR *SOFTWARE* CON TIA PORTAL

En TIA PORTAL se dispone de las mismas funciones que se han visto, pero en este caso se denominan instrucciones avanzadas. Vamos a ver dónde podremos encontrar las funciones que se han visto para STEP7 V5.5. Los parámetros son los mismos, por lo que se puede escribir el mismo programa que se ha hecho en el apartado anterior.

En la Figura 151 se indica dónde localizar la función SFC 28, llamada «SET_TINT» y la SFC 30, llamada «ACT_TINT». Están en la carpeta **Alarmas** de **Instrucciones avanzadas** del catálogo de instrucciones. El parámetro SDT que hay que incluir en la función SFC28 denominado #OB1_DATE_TIME también se encuentra en el OB1 de TIA PORTAL.

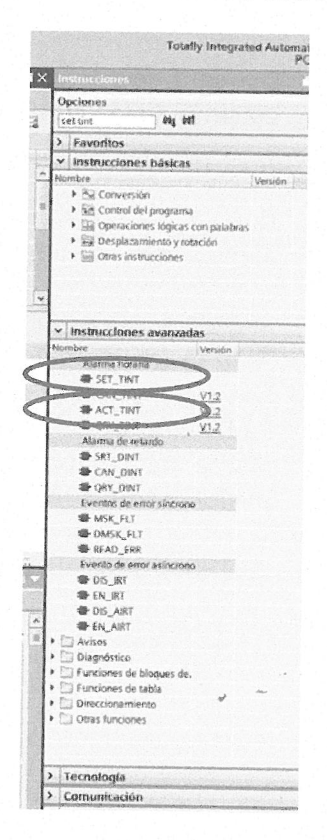


Figura 151

La función FC3 se encuentra en la carpeta **Fecha y hora** de las instrucciones avanzadas, como se puede comprobar en la Figura 152. Esta función se denomina **T_COMBINE**.

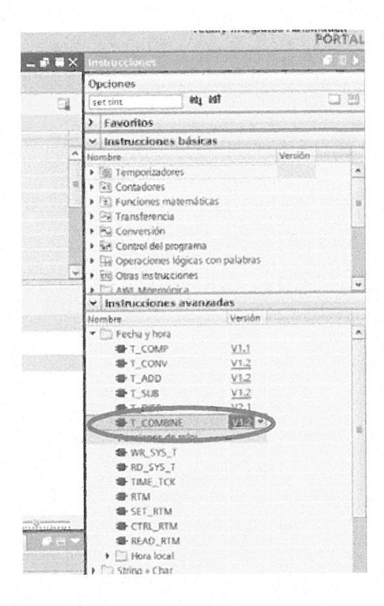


Figura 152

ALARMA DE RETARDO EN STEP7 V5.5

Este tipo de alarma hace que se ejecute una acción (un programa en el OB 20) un tiempo después de que se produzca el evento, es decir, la activación de la alarma. Hay que llamarla desde el OB1 y se debe hacer uso de la SFC 32.

Desde el OB1 se llama a la SFC 32 y es en este momento cuando empieza a contar el retardo hasta la ejecución del OB 20.

PARÁMETRO	TIPO DE DATO	SIGNIFICADO
OB_NR	INT	Número del OB que se arrancará al transcurrir el tiempo de retardo (OB 20 a OB 23). En los S7 300 solo OB 30.
DTIME	TIME	Valor del retardo (1 a 60000 ms). Puede establecer valores temporales mayores utilizando, por ejemplo, un contador en un OB de alarma de retardo.
SIGN	WORD	Signo que, al llamar el OB de alarma de retardo, aparece en la información de eventos de arranque del OB.
RET_VAL	INT	Si ocurre un error al procesar la función del sistema, el parámetro actual de RET_VAL contiene un código de error.

Esta alarma se utiliza como temporizador. Los temporizadores tienen un valor máximo de temporización de 9990 segundos y, si se desean tiempos mayores, habría que realizar combinaciones de temporizadores. Con la alarma el tiempo máximo es de 60 segundos. Dentro del OB 20 se puede introducir un temporizador, por lo que se podrá aumentar el tiempo máximo de dicho temporizador.

Una vez se activa la SFC 32, comienza la temporización. Se activa al soltar el pulsador que la activa, es decir, al pasar de 1 a 0.

Un ejemplo para utilizar la alarma de retardo OB 20:

Se hará lo mismo que para la alarma horaria. En el OB 20 se activará la salida A0.0 y en el OB1 se borrará esa salida con E0.0. La activación del retardo se hace con E0.1, que arrancará la SFC 32. El tiempo de retardo hay que introducirlo en el tipo TIME (no S5TIME) y se introduce con T#.

PROGRAMA OB 1

```
U E0.0
R A0.0

U E0.1
SPB L1

BEA

L1:CALL  "SRT_DINT"
     OB_NR  :=20
     DTIME  :=T#5S
     SIGN   :=MW40
     RET_VAL:=MW50
```

ALARMA DE RETARDO EN TIA PORTAL

Lo primero que se debe hacer es crear el OB20, tal como se hizo ya con el OB10. En la Figura 153 se puede comprobar cómo se genera.

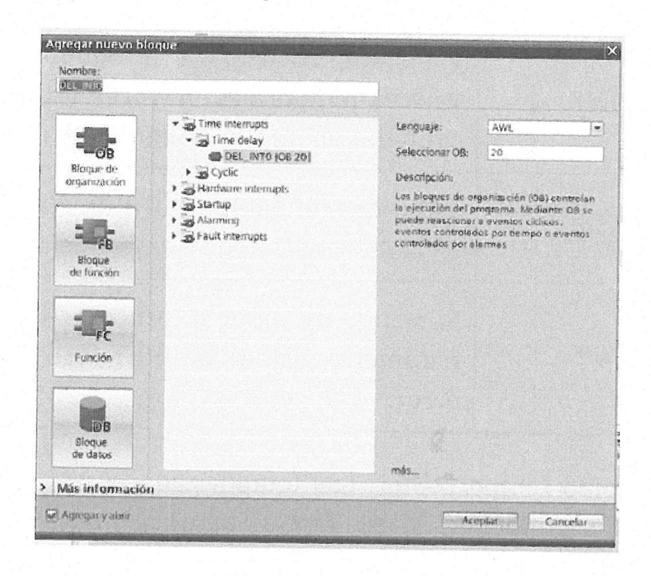


Figura 153

La función SFC32 se encuentra en la carpeta **Alarma**, según se ve en la Figura 154. Aquí se denomina **SRT_DINT**.

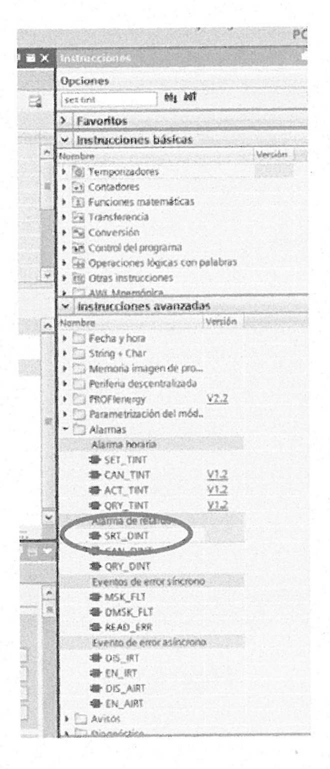


Figura 154

ALARMA CÍCLICA STEP7 V5.5

Las alarmas cíclicas se repiten de una forma continuada cada cierto tiempo. Se utilizan los OB 30 a OB 38. En los S7 300 solo se dispone del OB 35. Se configuran únicamente por *hardware.* Hay que hacerlo en la pantalla de propiedades de la CPU, tal como se ve en la Figura 155.

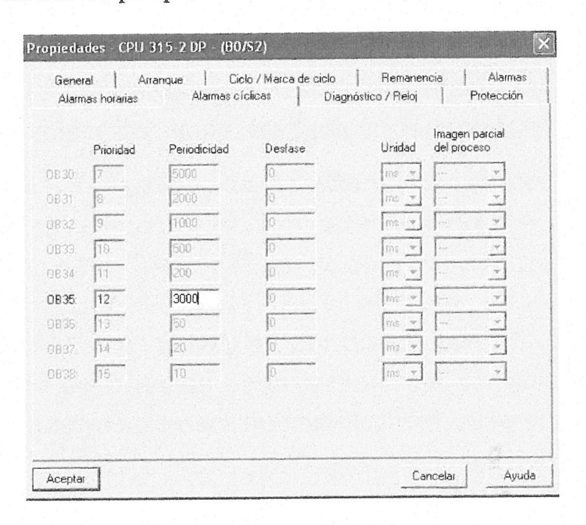


Figura 155

El tiempo se introduce en milisegundos. Una vez cambiado el tiempo, se debe compilar y enviar al PLC.

ALARMA CÍCLICA TIA PORTAL

Lo único que se debe hacer en TIA PORTAL es sacar el OB35 según aparece en la Figura 156.

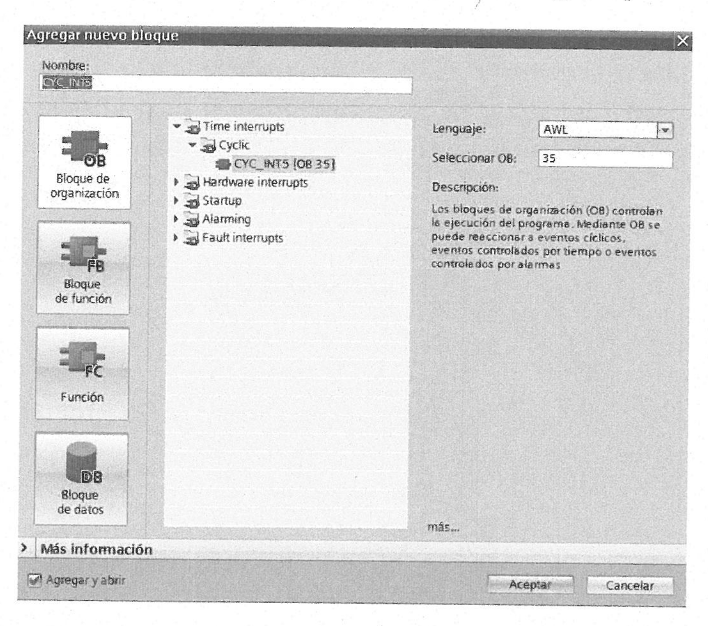


Figura 156

14. AUTÓMATA S7-1500

INTRODUCCIÓN

En los temas anteriores ya se ha hecho mención al PLC S7 1500. Se ha visto el formato exterior y su configuración. Respecto al *software*, se ha dicho que todas las órdenes del autómata S7 300 se pueden ejecutar en el S7 1500, pero es evidente que tiene otras instrucciones que lo hacen más potente. Una de sus características es que facilita el uso de la estandarización en la programación.

En este tema se van a estudiar algunas de las características que lo hacen diferencian del S7 300. El paso del autómata S7 300 al S7 1500 no debe ser un problema para nadie que tenga conocimientos de programación AWL en S7 300.

Algunos aspectos de tipo *hardware* del S7 1500 ya se han visto en los temas anteriores, así que aquí se hará más énfasis en la parte de *software*.

INTERCAMBIO DE DATOS EN AWL

Uno de los cambios importantes en el PLC 1500 es el uso de su memoria y, por lo tanto, el intercambio de datos. En el 1500 ya no hay acumuladores ni ningún otro tipo de registro en el procesador. Sin embargo, este autómata puede ejecutar todas las órdenes que ejecuta un S7 300. Esto se entiende porque el PLC 1500 puede emular los registros del PLC 300, es decir, acumuladores, palabra de estado, etc. De esta forma, se pueden compatibilizar las instrucciones del PLC 300.

Pero la forma óptima de trabajar con este nuevo PLC es a través de los parámetros de la interfaz de cada bloque, variables del PLC (mediante la **Tabla de variables**) o bloques de datos. Si, aun conociendo este hecho, se desean utilizar registros como el acumulador, hay que saber que la ejecución de los programas se ralentiza.

En la guía de Siemens *STEP 7* Professional *V13 SP1*, en el apartado *Intercambio de datos en AWL* para el PLC 1500, establece las siguientes indicaciones en lo que respecta a los registros:

- ✓ Los contenidos de los registros, de los acumuladores y de la palabra de estado solo están disponibles en segmentos AWL. Si un segmento KOP o FUP sigue a un segmento AWL, desde el segmento KOP o FUP no se podrá acceder a los contenidos de registro que antes se habían colocado en AWL. No obstante, en un segmento AWL posterior los contenidos de registro vuelven a estar disponibles. El bit RLO es una excepción: al realizar un cambio de lenguaje de programación se pone en «indefinido» y en los siguientes segmentos ya no estará disponible, aunque se vuelva programar en AWL.
- ✓ Los valores de los registros, de los acumuladores y de la palabra de estado no se transfieren a los bloques llamados. La única excepción son las instrucciones «CC» y «UC». Si utiliza «CC» o «UC», para transferir parámetros al bloque llamado a través de registros, de la palabra de estado o de acumuladores, **hay que activar la opción «Alimentación de parámetros a través de registros" en las propiedades del**

bloque llamado. Esta opción solo está disponible para bloques AWL con acceso estándar y cuando el bloque no puede tener parámetros formales. Si esta opción está activada, los contenidos de registro pueden transferirse entre bloques. También en este caso el bit RLO es una excepción: al salir del bloque se pone en «indefinido» y ya no estará disponible tras una llamada de bloque.

- ✓ Después de cada acceso a un bloque de datos indicando una dirección totalmente especificada (p. ej., %DB10.DBW10), el registro de bloques de datos DB se pone a «0». Un acceso posterior parcialmente especificado provoca un error durante la compilación.
- ✓ Para transferir una información de error al bloque invocante, se puede utilizar el bit BR. La información de error primero se debe guardar en el bloque llamado con la instrucción «SAVE» en el bit BR. A continuación se puede leer el bit BR en el bloque invocante.
- ✓ Si se direcciona simbólicamente un parámetro formal local desde la interfaz de bloque en S7-1500 (p. ej., con la instrucción L #myIn), siempre se accede al bloque de datos que se ha indicado como instancia en la llamada del bloque. Las instrucciones OPNDI, L AR2, +AR2, TDB y TAR cambian el contenido del registro DI o de direccionamiento, pero los registros ya no se evalúan en el direccionamiento de parámetros formales locales.

BLOQUE DE PROGRAMA OB1

El OB1 es el bloque principal donde se debe escribir el programa. El lenguaje que se va a utilizar aquí es AWL. Para seleccionarlo, se puede pulsar en **Main [OB1]** con el botón derecho y en la opción **Cambiar lenguaje de programación** e indicar el deseado. Esto es válido para los PLC de la serie S7 300/400, pero no para los S7 1500, ya que la opción AWL sale desactivada. Para este caso se debe borrar el **Main [OB1]** que sale por defecto y **Agregar nuevo bloque.** Se agrega un nuevo OB1 y desde allí se puede seleccionar el lenguaje AWL. Las figuras siguientes aclaran estas situaciones. la Figura 157 para un PLC 1500 y la Figura 158 para un PLC 300/400.

La Figura 159 indica cómo cambiar el lenguaje de programación desde la creación de un OB1 nuevo.

Los PLC de la serie S7 300/400 y S7 1500 pueden programarse en AWL-KOP-FUP, los de la serie S7 1200 KOP-FUP y todos los demás pueden hacerlo en SCL.

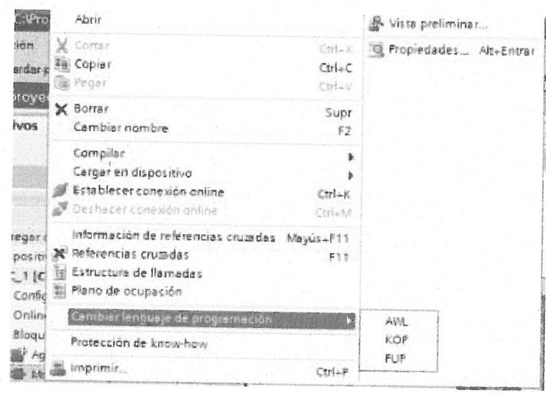


Figura 157

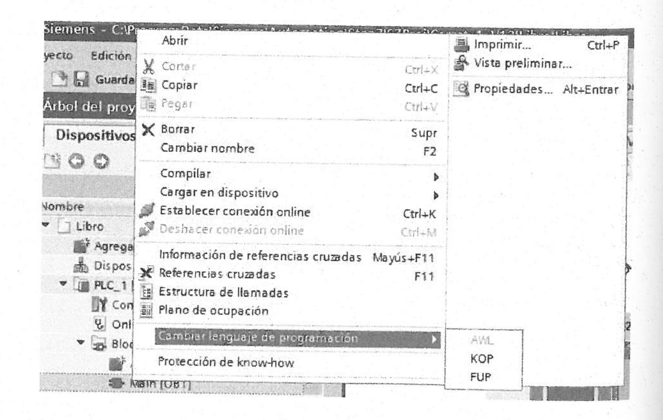


Figura 158

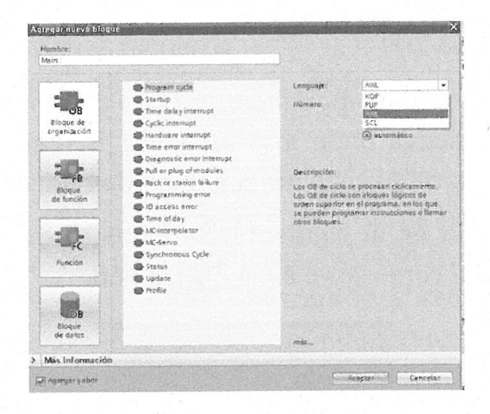


Figura 159

Las instrucciones se pueden escribir en el nemónico inglés o en el alemán. Por defecto, toma el idioma inglés pero son muchos los programadores de PLC de Siemens que, por costumbre, utilizan el alemán. Para cambiarlo hay que ir a:

Opciones/Configuración/Programación PLC/General/Otros ajustes/Nemónicos: Alemán.

TEMPORIZADORES IEC

La norma IEC 61131 es una norma de estandarización que se aplica a los autómatas programables, en cuanto al *software* y al *hardware*. Tiene varios apartados y concretamente el tercero hace referencia a la programación de los autómatas (IEC 61131-3). Para obtener más información, se puede consultar la siguiente página de internet:

http://www.plcopen.org/pages/tc1_standards/

Es un intento por conseguir la estandarización de la programación de autómatas programables, que no dependa del fabricante del autómata.

El PLC S7 1500 está basado en esta norma y sigue los criterios de la IEC 61131-3.

En cuanto a los temporizadores existen cuatro tipos basados en esta norma:

TP: Impulso / TON: Retardo al conectar

TOF: Retardo al desconectar /TONR: Acumulador de tiempo.

Cada uno de estos temporizadores usa un FB y un DB asociado. Para utilizarlos, basta con hacer la llamada al FB correspondiente, seleccionando un DB de instancia. Este DB dispone de unos datos que son los parámetros del FB y que, a su vez, son los parámetros que se deben utilizar para hacer uso del temporizador. Se pueden hacer las llamadas que se crean necesarias desde cualquier bloque (OB1, FC, FB).

Estos temporizadores no ocupan memoria fija en el PLC, y cada vez que se utilizan, se dispone de esa memoria. No ocurre igual en los temporizadores usados en las versiones de STEP 7 para el PLC S7 300/400, que sí disponen de una zona de memoria fija.

TEMPORIZADOR TP (IMPULSO)

El comportamiento de este tipo de temporizador es igual al tipo **V** (impulso prolongado) de la serie 300/400. La salida se activa nada más arrancar el temporizador y no es necesario mantener la acción que lo arranca. Al concluir el tiempo de retardo, la salida se pone a cero.

El procedimiento para crearlo es el siguiente. Se arrastra la orden desde **Instrucciones básicas** en la parte derecha, tal como se ve en la Figura 160.

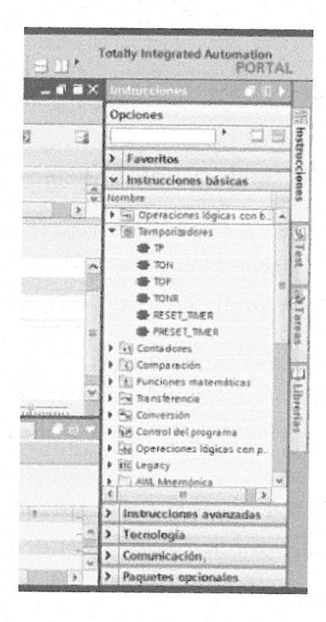
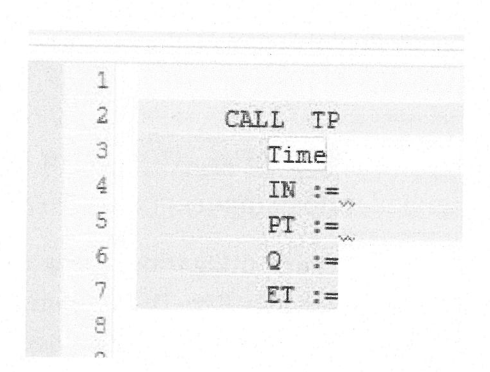


Figura 160

Al colocar la instrucción en el OB1, se muestran los parámetros del FB, según se indica en la Figura 161. Ahora es necesario un DB asociado a este FB, que se creará cuando se seleccione el tipo de dato del tiempo del temporizador (Figura 162).



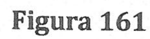


Figura 161

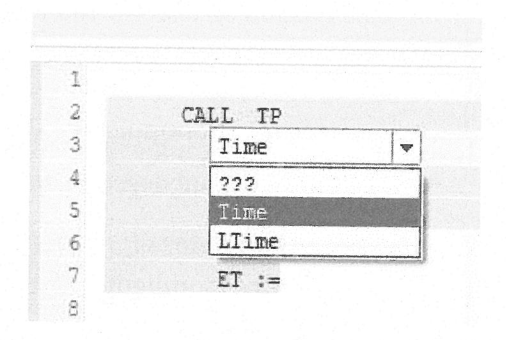


Figura 162

Estos datos de tiempo pueden ser del tipo Time o LTime.

El contenido de un operando del tipo TIME se interpreta como milisegundos. La representación contiene especificaciones de días (d), horas (h), minutos (m), segundos (s) y milisegundos (ms). No es necesario indicar todas las unidades de tiempo. Por ejemplo, T#5h10s es válido. Si se indica solo una unidad, el valor absoluto de días, horas y minutos no

podrá exceder los límites superiores ni inferiores. Si se indica más de una unidad de tiempo, el valor correspondiente no podrá exceder 24 días, 23 horas, 59 minutos, 59 segundos o 999 milisegundos.

El contenido de un operando del tipo LTIME se interpreta como nanosegundos. La representación contiene especificaciones de días (d), horas (h), minutos (m), segundos (s), milisegundos (ms), microsegundos (us) y nanosegundos (ns). No es necesario indicar todas las unidades de tiempo. Por ejemplo, LT#5h10s es válido. Si se indica solo una unidad, el valor absoluto de días, horas y minutos no podrá exceder los límites superiores ni inferiores. Si se indica más de una unidad de tiempo, el valor correspondiente no podrá exceder 106751 días, 23 horas, 59 minutos, 59 segundos, 999 milisegundos, 999 microsegundos o 999 nanosegundos.

Cuando se selecciona el tipo de dato, Time o LTime aparece la ventana para seleccionar el DB según se aprecia en la Figura 163. Se puede cambiar el nombre del DB.

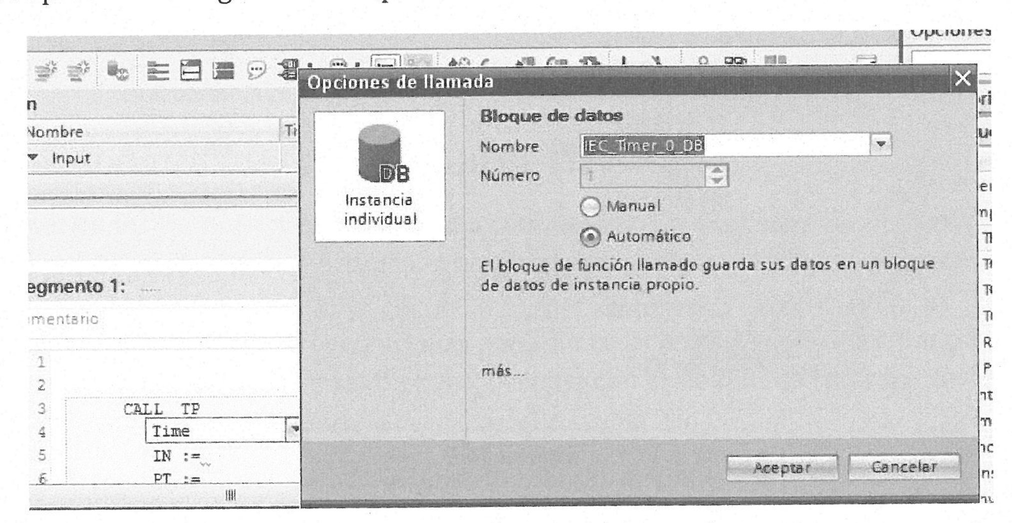


Figura 163

Los parámetros que se deben introducir son:

IN: la entrada que arranca el temporizador.

PT: el valor de consigna del tiempo que se desea temporizar, colocado después de T#.

Q: la salida que se va a activar junto al temporizador.

ET: es el registro donde se carga el tiempo colocado en PT y donde se puede ver cómo va aumentando hasta llegar al tiempo seleccionado en PT. Si no se indica ninguna variable, funcionará igual. Se puede visualizar el estado del temporizador colocando ET en la tabla de observación de variables. Si se indica algo, debe ser una doble palabra o real.

Ejemplo: en este ejemplo se ha cambiado el nombre por defecto del DB por "T1".

```
        CALL  TP , "T1"                        %DB2
          Time
          IN :="Arranque"                      %E0.0
          PT :=T#0.5d2h300s150ms               T#0.5d2h300s1...
          Q  :="Salida"                        %A0.0
          ET :="Tiempo_Actual"                 %MD20
```

Figura 164

El temporizador se puede parar con la orden de reseteo: RESET_TIMER.

```
        CALL  TP , "T1"                        %DB2
          Time
          IN :="Arranque"                      %E0.0
          PT :=T#0.5d2h300s150ms               T#0.5d2h300s1...
          Q  :="Salida"                        %A0.0
          ET :="Tiempo_Actual"                 %MD20

        U     "Parar T1"                       %E0.1
        SPB   l1
        BEA

  l1:   CALL  RESET_TIMER
          IEC_Timer
          TIMER :="T1"                         %DB2
```

Figura 165

TEMPORIZADOR TON (RETARDO AL CONECTAR)

Este tipo de comportamiento es igual al tipo **E** de la serie S7 300/400. Desde que se arranca el temporizador tarda un tiempo hasta que se conecta la salida. Es necesario mantener la acción que lo ha arrancado durante todo el tiempo de temporización. El procedimiento de utilización es idéntico al que se ha visto para el tipo TP.

Ejemplo:

```
        CALL  TON , "T1"                       %DB2
          Time
          IN :="Arranque"                      %E0.0
          PT :=T#5s                            T#5s
          Q  :="Salida"                        %A0.0
          ET :="Tiempo_Actual"                 %MD20
```

Figura 166

TEMPORIZADOR TOF (RETARDO AL DESCONECTAR)

En este caso, el comportamiento de este temporizador es como el tipo **A** en los PLC S7 300/400. Al pulsar la activación, la salida se activa pero el temporizador no arranca hasta soltar (en el paso de 1 a 0). El procedimiento es el mismo al visto en los casos anteriores.

Ejemplo:

```
CALL  TOF , "T1"                    %DB2
   Time
   IN :="Arranque"                  %E0.0
   PT :=t#5s                        t#5s
   Q  :="Salida"                    %A0.0
   ET :="Tiempo_Actual"             %MD20
```

Figura 167

TEMPORIZADOR TONR (ACUMULADOR DE TIEMPO)

Este temporizador no tiene ningún parecido con los del tipo S7 300/400. El comportamiento es similar al TON: desde que se arranca el temporizador tarda un tiempo hasta que se conecta la salida. Es necesario mantener la acción que lo ha arrancado todo el tiempo de temporización porque en caso contrario, se para. La característica diferenciadora está en que cuando se vuelve a activar, después de parado, sigue con el tiempo de retardo que tenía antes de la parada. Para pararlo y llevarlo a cero, hay que resetearlo.

Además de las entradas de los otros tipos de temporizadores, este tiene una entrada para resetearlo. Ejemplo:

```
CALL  TONR , "T1"                   %DB2
   Time
   IN :="Arranque"                  %E0.0
   R  :="Reiniciar T1"              %E0.1
   PT :=t#5s                        t#5s
   Q  :="Salida"                    %A0.0
   ET :=
```

Figura168

TEMPORIZADORES CON MULTINSTANCIAS

Como se ha comprobado en los ejemplos anteriores, cada vez que se utiliza un temporizador, se hace uso de un DB. Esto puede generar problemas de memoria y de estructuración de la misma, así como de organización del propio programa (no problemas de funcionamiento). En cada DB se tendrán los datos/parámetros de cada uno de los temporizadores. Si se utilizan 20 temporizadores, habrá 20 DB.

Hay otra posibilidad de trabajar, que es utilizando multiinstancias. De este modo solo se tendrá un DB donde estarán centralizados todos los parámetros de todos los temporizadores utilizados. Con esto se gana en organización, ya que, aunque no se puede simular, sí se puede visualizar y ver el estado online de todos los parámetros de los temporizadores desde un único DB.

Para usar multiinstancias con los temporizadores se debe seguir el siguiente procedimiento:

1. Se crea un FB.
2. Dentro de ese FB se hacen las llamadas a los temporizadores necesarios. Se debe grabar.
3. Se coloca el tipo de variable en ???, Time o LTime; cuando solicite el DB, se selecciona **multiinstancia** y ahora (desde multiinstancia, no antes) se asigna un nombre a esa instancia del temporizador. Se graba.
4. Si se hace otra llamada para crear otro temporizador, se debe hacer lo mismo e indicar otro nombre para la nueva instancia. Hay que volver a grabar.
5. En el OB1 se hace la llamada al FB. Al poner **Call FB1** y pulsar **Enter**, se solicita crear un DB (de instancia).
6. Se debe activar el icono para solucionar las posibles incoherencias, si las hay.
7. Al cargar en el PLC, si sale la opción **Reinicialización de bloques de datos**, hay que cambiar la opción de **Ninguna acción** a **Inicializar**, tal como se aprecia en la Figura 169.

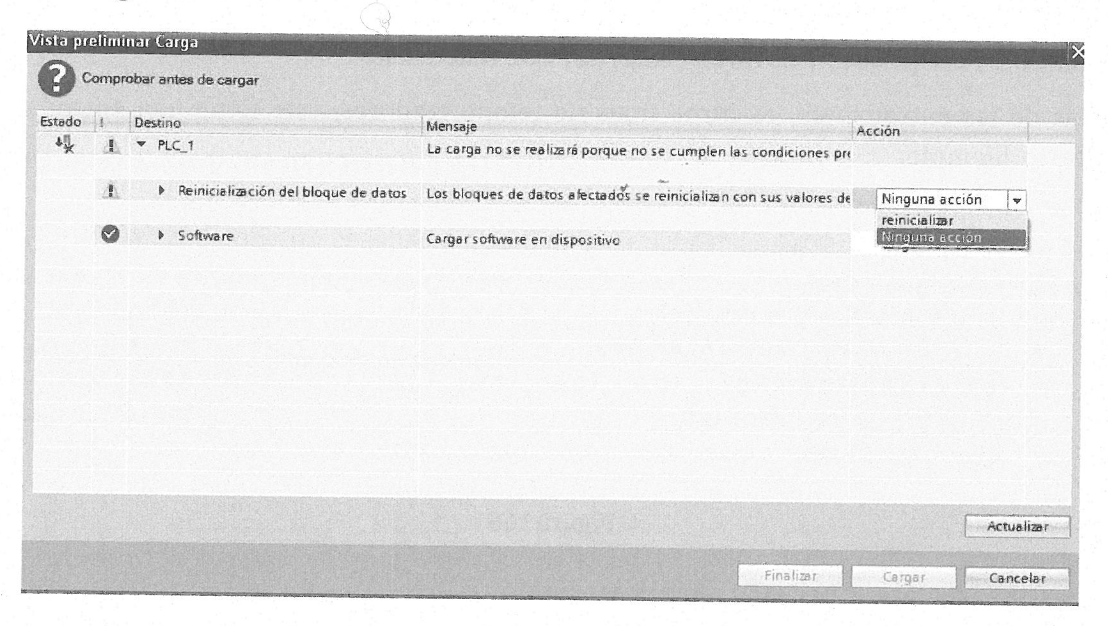


Figura 169

En el siguiente ejemplo se puede ver cómo queda cada uno de los bloques, el OB1, el FB1 y su DB1 de multiinstancia.

OB1

```
CALL  "Bloque_1", "Bloque_1_DB"                    %FB1, %DB1
```

FB1: en este ejemplo se han utilizado tres temporizadores: uno TP, otro TON y otro TOF.

```
CALL  #T1_TP
   Time
   IN :="Arranque T1"                              %E0.0
   PT :=T#3S                                       T#3S
   Q  :="Salida1"                                  %A0.0
   ET :=

CALL  #T2_TON
   Time
   IN :="Arranque T2"                              %E0.1
   PT :=T#5s                                       T#5s
   Q  :="Salida2"                                  %A0.1
   ET :=

CALL  #T3_TOF
   Time
   IN :="Arranque T3"                              %E0.2
   PT :=t#8s                                       t#8s
   Q  :="Salida3"                                  %A0.2
   ET :=
```

DB1

	Nombre	Tipo de datos	Valor de arranq...	Valor de observación	Remanen
1	Input				
2	Output				
3	InOut				
4	Static				
5	T1_TP	TP_TIME			
6	ST	Time	T#0ms	T#13D_14H_30M_...	
7	PT	Time	T#0ms	T#3S	
8	ET	Time	T#0ms	T#0MS	
9	RU	Bool	false	FALSE	
10	IN	Bool	false	FALSE	
11	Q	Bool	false	FALSE	
12	T2_TON	TON_TIME			
13	ST	Time	T#0ms	T#13D_14H_31M_...	
14	PT	Time	T#0ms	T#5S	
15	ET	Time	T#0ms	T#0MS	
16	RU	Bool	false	FALSE	
17	IN	Bool	false	FALSE	
18	Q	Bool	false	FALSE	
19	T3_TOF	TOF_TIME			
20	ST	Time	T#0ms	T#13D_14H_31M_...	
21	PT	Time	T#0ms	T#8S	
22	ET	Time	T#0ms	T#8S	
23	RU	Bool	false	FALSE	
24	IN	Bool	false	FALSE	
25	Q	Bool	false	FALSE	

CONTADORES IEC

Como ocurre con los temporizadores, también en los contadores se puede utilizar el *software* para los contadores de los PLC de la serie S7 300/400. Pero existen otro tipo de contadores, que son los tipos IEC. Hay que recordar que estos no tienen un espacio de memoria reservado en el PLC y, por lo tanto, el límite del número de contadores dependerá de la propia memoria.

En los contadores IEC se utiliza igualmente un DB de instancia por cada contador. Como se ha visto en los temporizadores, en los contadores igualmente se puede optimizar su utilización haciendo uso de la multiinstancia, armonizando y optimizando mejor la memoria.

En IEC existen contadores de subida, bajada y subida/bajada. Estos contadores pueden pasar del valor máximo y también tener números negativos, descendiendo de cero. El límite de contaje depende del tipo de la variable utilizada (Int, DInt, UsInt...).

CONTADOR ASCENDENTE (CTU)

La instrucción **Contador ascendente** incrementa el valor del registro contador. Cuando el estado lógico de la entrada del contador cambia de 0 a 1 (flanco de señal ascendente), se ejecuta la instrucción y el valor de contaje actual del registro del contador se incrementa en uno. El valor de contaje se incrementa cada vez que se detecta un flanco de señal ascendente.

Para utilizar este contador se debe arrastrar la orden desde contadores en instrucciones básicas. Como ya se hacía con los temporizadores, se debe seleccionar el tipo de datos que se va a utilizar (Int, DInt.....) situándose sobre los tres interrogantes ("???"). En este momento se selecciona el DB de instancia a este FB y se puede modificar el nombre.

Esta instrucción es un FB que dispone de un DB asociado con los siguientes parámetros:

CU: es la entrada del contador. Por cada flanco positivo, se incrementa el contador.

R: es la entrada para poner a cero el contador. Con un uno se reinicializa el contador.

PV: es el valor máximo que se desea para el contador. Cuando el registro contador (CV) alcanza al valor colocado aquí, la salida del contador (Q) se pone a uno. Si lo sobrepasa, la salida sigue a uno. Cuando se llegue al valor máximo del contador, su registro (CV) seguirá incrementándose. El valor máximo que se puede indicar aquí dependerá del tipo de variable seleccionado en "???".

Q: este parámetro de salida se pone a uno cuando CV> = PV. Es un indicador de haber alcanzado el valor establecido en el contador con PV.

CV: en este parámetro se tiene el valor actual del contador. El tipo de dato dependerá del tipo de variable seleccionado en "???".

En la lógica del programa, la instrucción «Contador ascendente» se debe llamar con la instrucción «Llamar bloque» (CALL).

Para evitar problemas con el contaje, el contador debe llamarse en una sola posición del programa.

A cada llamada de la instrucción «Contador ascendente» se le debe asignar un contador IEC en el que se guarden los datos de la instrucción.

Ejemplo:

```


      CALL  CTU , "Z1_UP"                         %DB1
         Int
         CU :="Entrada1"                          %E0.0
         R  :="Reset_Z1"                          %E0.1
         PV :=5                                   5
         Q  :="Salida_Z1"                         %A0.0
         CV :="Estado_actual_Z1"                  %MW1
```

CONTADOR DESCENDENTE (CTD)

La instrucción «Contador descendente» decrementa el valor del registro contador. Cuando el estado lógico de la entrada del contador cambia de 0 a 1 (flanco de señal ascendente), se ejecuta la instrucción y el valor de contaje actual del registro del contador se decrementa en uno. El valor de contaje se decrementa cada vez que se detecta un flanco de señal ascendente.

Para utilizar este contador, se debe arrastrar la orden desde contadores en instrucciones básicas. Como ya se hacía con los temporizadores, se debe seleccionar el tipo de datos que se va a utilizar (Int, DInt.....) y situarlos sobre los tres interrogantes ("???"). En este momento se selecciona el DB de instancia a este FB. Se puede modificar el nombre.

Esta instrucción es un FB que dispone de un DB asociado con los siguientes parámetros:

CD: es la entrada del contador. Por cada flanco positivo se decrementa el contador.

LD: es la entrada para poner a un valor deseado el contador. Con un uno se carga el contador a ese valor.

PV: es el valor con el que se desea cargar el contador cuando se active LD.

Q: este parámetro se pondrá a uno cuando el contador sea cero o menor, cuando CV <= 0.

CV: en este parámetro se tendrá en valor actual del contador. El contador puede pasar por debajo de cero (valores negativos).

En la lógica del programa, la instrucción «Contador descendente» se debe llamar con la instrucción «Llamar bloque» (CALL). Ejemplo:

```
      CALL  CTD , "Z2_DOWN"                       %DB2
         Int
         CD :="Entrada_Z1DOWN"                    %E0.2
         LD :="Cargar_Z2"                         %E0.3
         PV :=20                                  20
         Q  :="Salida_Z2"                         %A0.1
         CV :="Valor_Actual_Z2"                   %MW3
```

Para evitar problemas con el contaje, el contador debe llamarse en una sola posición del programa.

A cada llamada de la instrucción «Contador descendente» se le debe asignar un contador CEI en el que se guarden los datos de la instrucción.

CONTADOR ASCEDENTE/DESCENDENTE (CTUD)

Los dos contadores anteriores se podrían utilizar sobre una misma instancia y así obtener un contador de subida y bajada. Pero se dispone de una utilidad mucho más cómoda: un contador IEC con los parámetros necesarios para que se pueda incrementar y decrementar el registro.

El proceso para utilizarlo es el mismo que para los dos anteriores. Es una conjunción de los dos contadores. Los parámetros de este contador son: CU, CD, R, LD, PV, QU, QD, CV, que son los mismos parámetros que se han enunciado en los contadores ascendentes y descendentes.

El parámetro QU es la salida para el contador en sentido ascendente, que se activa cuando CV >= PV. El parámetro QD es la salida para el contador en sentido descendente, es decir, cuando CV <=0.

El valor de PV es el valor para el que el contador pone la salida QU a uno cuando CV >= PV. PV también es el valor con el que se cargará el contador al activar LD. Es decir, cuando LD es uno, CV se pone con el valor de PV.

Ejemplo:

```
      CALL  CTUD , "Z3_U_D"                       %DB3
         Int
         CU :="Entrada_UP"                        %E0.0
         CD :="Entrada_DOWN"                      %E0.2
         R  :="Reset_Z3"                          %E0.1
         LD :="Cargar_Z3"                         %E0.3
         PV :=20                                   20
         QU :="Salida_Z3_UP"                      %A0.0
         QD :="Salida_Z3_DOWN"                    %A0.1
         CV :="Valor_Actual_Z3"                   %MW4
```

Con las entradas E0.0 y E0.2 el contador se incrementa o decrementa respectivamente. El valor inicial del contador será 20. Se podrá volver a cargar en cualquier momento accionando la entrada E0.3. Cuando el contador llegue a 20 (desde 20 en adelante), la salida QU se pondrá a uno. Por otra parte, cuando el contador llegue a cero, la salida QD se pondrá a uno. En CV se puede ver el valor actual del contador.

CONTADORES CON MULTINSTANCIAS

La utilización de contadores en el estándar IEC incluye un DB de instancia por cada contador. Como ya ocurría con los temporizadores, el uso de una gran cantidad de contadores puede hacer incómoda su visualización online. Por ello, se puede utilizar multiinstancias, tal como ocurre en los temporizadores. De este modo se tendrán todos los contadores en un mismo DB. Desde ese mismo DB, puesto online, se pueden observar todos los contadores.

El procedimiento es el mismo que para los temporizadores, por lo tanto se omite el proceso de realización para los contadores.

EJERCICIO DE TEMPORIZADORES Y CONTADORES

Se disponen de tres motores que deben realizar la siguiente secuencia:

Primero debe ponerse en funcionamiento el Motor1 mediante el pulsador PM1. Arranca al cabo de 5 segundos de ser accionado PM1 y se para 8 segundos después. A continuación, y accionando 5 veces el pulsador PM2, arranca el Motor2. Pulsando 10 veces más PM2, arranca

el Motor3. Este motor se para accionando tres veces más el pulsador PM2. El Motor2 se para cuando el contador desciende hasta cero mediante el pulsador PM3.

Para la realización de este ejercicio se va a utilizar grafcet.

Las variables usadas son las siguientes:

ENTRADAS		SALIDAS		MARCAS	
pm1	E0.0	motor1	A0.0	Etapa0	M0.0
pm2	E0.1	motor2	A0.1	Etapa1	M0.1
pm3	E0.2	motor3	A0.2	Etapa2	M0.2
				Etapa3	M0.3
				Etapa4	M0.4
				Etapa5	M0.5
				Etapa6	M0.6
				Etapa7	M0.7
				ContadorZ1_10	M10.0
				ContadorZ1_13	M13.0
				Marca_30.0	M30.0
				Marca_30.1	M30.1

GRAFCET

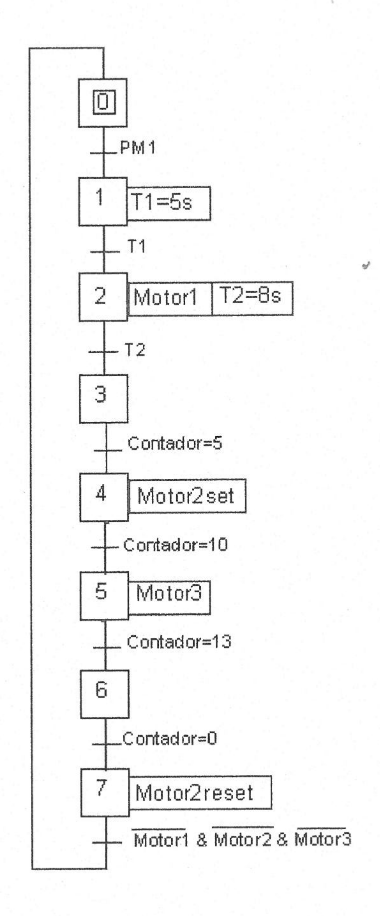


Figura 170

PROGRAMA

OB1

//TRANSICIONES	//ACCIONES

U "pm1"
U "Etapa0"
R "Etapa0"
S "Etapa1"
 	
U "T1".Q
 	
U "Etapa1"
R "Etapa1"
S "Etapa2"
 	
U "T2".Q
U "Etapa2"
R "Etapa2"
S "Etapa3"
 	
U "Z1".QD
U "Etapa3"
R "Etapa3"
S "Etapa4"
 	
U
"ContadorZ1_10"
U "Etapa4"
R "Etapa4"
S "Etapa5"
 	
U
"ContadorZ1_13"
U "Etapa5"
R "Etapa5"
S "Etapa6"
 	
U "Z1".QD
U "Etapa6"
R "Etapa6"
S "Etapa7"
 	
UN "motor1"
UN "motor2"
UN "motor3"
U "Etapa7"
R "Etapa7"
S "Etapa0"
	CALL TON, "T1"
	time_type:=Time
	IN :="Etapa1"
	PT :=T#5s
	Q :=
	ET :=
	
	CALL TON, "T2"
	time_type:=Time
	IN :="Etapa2"
	PT :=T#8s
	Q :=
	ET :=
	
	U "Etapa2"
	= "motor1"
	
	U "Etapa4"
	S "motor2"
	
	U "Etapa5"
	= "motor3"
	
	U "Etapa7"
	R "motor2"
	
	//El pulsador PM2 solo debe incrementar el contador en las etapas 3, 4 y 5.
	O "Etapa3"
	O "Etapa4"
	O "Etapa5"
	U "pm2"
	= "Marca30.0"
	
	//El pulsador PM3 solo debe incrementar el contador en la etapa 6
	U "pm3"
	U "Etapa6"
	= "Marca30.1"
	
	//Las marcas activadas anteriormente son las que incrementan (M30.0) o decrementan (M30.1) el contador Z1.
	CALL CTUD , "Z1"
	value_type:=Int
	CU :="Marca30.0"

	CD :="Marca30.1"
	R :=
	LD :=
	PV :=5
	QU :=
	QD :=
	CV :=
	
	//Comparador de Z1 es mayor o igual a 10
	L "Z1".CV
	L 10
	>=I
	= "ContadorZ1_10"
	//Comparador de Z1 es mayor o igual a 13
	L "Z1".CV
	L 13
	>=I
	= "ContadorZ1_13"

OB100

```
SET
S   "Etapa0"
```

COMPARADORES

Las instrucciones de comparación para el PLC S7 1500 son las mismas que las utilizadas en Step 7 para S7 300/400.

Hay otras instrucciones de comparación de tipos de datos con los punteros (Variant). Estas sirven para consultar si el contenido de las variables en las que apunta el puntero son del mismo tipo que otra variable cualquiera.

Las instrucciones de comparación más útiles son las que ya existían en la serie S7 300/400. Con ellas se pueden comparar números enteros, reales y dobles palabras. Si se cumple la comparación, el RLO se pone a uno.

Existe una comparación muy útil que puede comparar tiempo, y fecha y hora. Esa instrucción es T_COMP, y se encuentra en **Fecha y hora** en **Instrucciones avanzadas.** Para realizar la comparación, los tipos de datos de las variables que se comparan deben ser iguales.

Los parámetros de esta instrucción son:

IN1: primer valor a comparar.

IN2: segundo valor a comparar.

OUT: valor de respuesta a la comparación. Se pone a uno si se cumple.

Los tipos de datos válidos en la comparación son: Date, Time, LTime, TOD (Time_Of_Day), LTOD (LTime_Of_Day), DT (Date_And_Time), DTL (Date_And_LTime), S5Time.

Existen varias posibilidades de comparación:

EQ: la salida (OUT) es uno si IN1 = IN2.

NE: la salida (OUT) es uno si IN1 ≠ IN2.

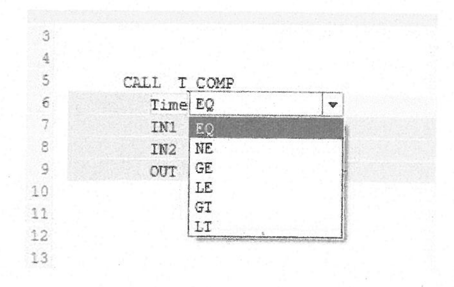
GE: la salida (OUT) es uno si IN1 >= IN2. También se puede leer como si IN1 es más reciente o igual que IN2.

LE: la salida (OUT) es uno si IN1 <= IN2. También se puede leer como si IN1 es más antiguo o igual que IN2.

GT: la salida (OUT) es uno si IN1 > IN2. También se puede leer como si IN1 es más reciente que IN2.

LT: la salida (OUT) es uno si IN1 < IN2. También se puede leer como si IN1 es más antiguo que IN2.

En la instrucción de comparar tiempo se debe seleccionar la opción requerida:



Ejemplo:

```
CALL TP , "IEC_Timer_0_DB"                        %DB2
   Time
   IN :="Entrada_UP"                              %E0.0
   PT :=T#20S                                     T#20S
   Q  :="Salida_Z3_UP"                            %A0.0
   ET :=



CALL T_COMP
   Time GE
   IN1 :="IEC_Timer_0_DB".ET
   IN2 :=t#5s                                     t#5s
   OUT :="Salida_Z3_DOWN"                         %A0.1
```

En este ejemplo se utiliza un temporizador IEC tipo TP de 20 segundos. La instrucción de comparación contrasta entre el tiempo del temporizador y 5 s, de forma que cuando el tiempo del temporizador sobrepase los 5 s, se activará la salida A0.1

MULTIINSTANCIAS CON TIA PORTAL Y PLC S7-1500

Acabamos de comprobar cómo se puede trabajar con los temporizadores y contadores de una forma más efectiva utilizando multiinstacias. Insistiremos sobre este importante asunto, pero de una forma general para cualquier aplicación, no solo en el uso de los temporizadores.

Ya se sabe que un FB siempre lleva asociado un DB. Hemos demostrado cómo el uso de multiinstancias optimiza la programación y el propio PLC. La sinopsis de la multiinstancia es que un FB llama a otros FB disponiendo de único DB de acceso multiinstacia. Dicho así, no se aprecian las ventajas del uso de esta forma de trabajar.

Simplificando, se podría de decir que hay un FB (general) que sería como una plantilla y los diferentes FB (individuales) que hacen llamadas al FB (general) serían las actualizaciones de las plantillas o las diversas aplicaciones de las plantillas. Cada uno de esos FB (individuales) están basados en la plantilla del FB (principal), pero con datos distintos. Esto resulta muy útil en recetas donde la base es la misma pero cambian diferentes valores.

Se va a realizar un ejemplo que aclarará su utilización. Se trata de una fábrica de bebidas que produce tres tipos de productos: néctar, zumo y refresco. Cada producto dispone de una cierta cantidad de sustancias que están depositadas en diferentes silos. Las sustancias son: azúcar, aditivos y concentrado. Esas sustancias caen a un concentrador donde un agitador las remueve para dejar el producto listo para su envasado. El agitador no entra en funcionamiento hasta que todos los productos han caído en el concentrador.

Lo que hace diferente a cada producto es la cantidad de sustancia que cae al concentrador y eso se calcula mediante el tiempo que se deja la puerta de la tolva abierta. En la tabla siguiente se especifican dichos tiempos.

	Azúcar	Aditivos	Concentrado	Agitador
NÉCTAR	5 s	6 s	4 s	7 s
ZUMO	4 s	2 s	10 s	3 s
REFRESCO	7 s	8 s	2 s	10 s

En el procedimiento se utilizarán multiinstacias. El proceso será:

1.º Crear el FB1 general; en él se colocarán el programa principal y los parámetros locales (valores formales) que serán en este caso las diferentes sustancias de cada producto. A esta FB1 se le llamará Arrancar_bebida. En este caso serán entradas y del tipo Time (T#).

2.º Crear un FB2 con las diferentes instancias (multiinstancias). Cada una de las instancias corresponderá a cada una de las bebidas: néctar, zumo y refresco. Para ello, en esta FB2 hay que crear tres parámetros estáticos (STATIC) y de tipo FB1 (Arrancar_bebida). De esa forma, cada instancia tendrá la forma de la plantilla, es decir, del FB1.

Una vez hecho esto, se realizarán las llamadas al FB1 con **multiinstancia** seleccionando la instancia deseada (néctar, zumo, refresco).

3.º Desde el OB1 hacer la llamada al FB1 y entonces crear el DB1 asociado de multiinstancia. Hasta ahora no había ningún DB creado y solo habrá un DB que será del FB1 con acceso multiinstancia desde FB2.

Se van a realizar los pasos aplicados al ejercicio a través de las pantallas.

Las variables globales declaradas en las variables del PLC son las que aparecen en la Figura 171.

Multiinstancias ▸ PLC_1 [CPU 1516-3 PN/DP] ▸ Variables PLC ▸ Tabla de variables estándar [61]

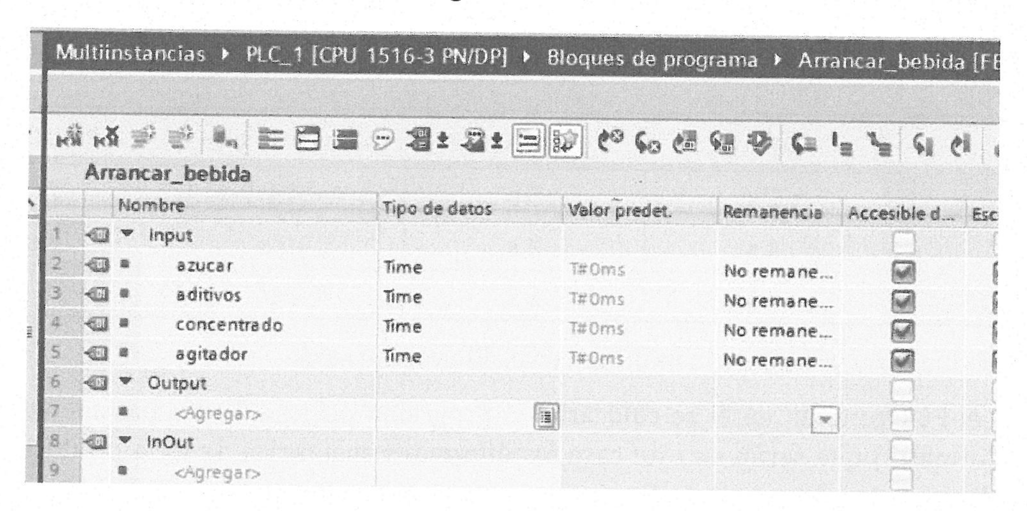
Variables | Constantes de usuario

Tabla de variables estándar

	Nombre	Tipo de datos	Dirección	Rema...	Acces...	Escrib...	Visibl...	Supervis
1	Activacion	Bool	%E0.0	☐	☑	☑	☑	
2	Selector_Nectar	Bool	%E1.0	☐	☑	☑	☑	
3	Selector_Zumo	Bool	%E1.1	☐	☑	☑	☑	
4	Selector_Refresco	Bool	%E1.2	☐	☑	☑	☑	
5	Tag_10	Bool	%E1.3	☐	☑	☑	☑	
6	Tolva_Azucar	Bool	%A0.0	☐	☑	☑	☑	
7	Tolva_Aditivos	Bool	%A0.1	☐	☑	☑	☑	
8	Tolva_Concentrado	Bool	%A0.2	☐	☑	☑	☑	
9	Agitador	Bool	%A0.3	☐	☑	☑	☑	
10	Memoria_activación	Bool	%M0.0	☐	☑	☑	☑	
11	Activacion_Agitador	Bool	%M0.1	☐	☑	☑	☑	
12	<Agregar>			☐	☑	☑	☑	

Figura 171

1.º En primer lugar hay que crear el FB1 (el principal o la plantilla). Los parámetros creados en esta FB1 son los representados en la Figura 172.



Multiinstancias ▸ PLC_1 [CPU 1516-3 PN/DP] ▸ Bloques de programa ▸ Arrancar_bebida [FB

Arrancar_bebida

	Nombre	Tipo de datos	Valor predet.	Remanencia	Accesible d...	Esc
1	▾ Input				☐	
2	azucar	Time	T#0ms	No remane...	☑	
3	aditivos	Time	T#0ms	No remane...	☑	
4	concentrado	Time	T#0ms	No remane...	☑	
5	agitador	Time	T#0ms	No remane...	☑	
6	▾ Output				☐	
7	<Agregar>				☐	
8	▾ InOut				☐	
9	<Agregar>				☐	

Figura 172

El programa del bloque de función FB1 (Arrancador_bebida) es el siguiente:

// Los temporizadores que mantiene la tolva abierta arrancan nada más activar la entrada de Activación (E0.0).

```
CALL TP , "t1"
Time
IN :="Activacion"
PT :=#azucar
```

```
Q :="Tolva_Azucar"
ET :=

CALL TP , "t2"
Time
IN :="Activacion"
PT :=#aditivos
Q :="Tolva_Aditivos"
ET :=

CALL TP , "t3"
Time
IN :="Activacion"
PT :=#concentrado
Q :="Tolva_Concentrado"
ET :=

// El agitador debe entrar en acción al cerrarse las tres tolvas

CALL TP , "t4"
Time
IN :="Activacion_Agitador"
PT :=#agitador
Q :="Agitador"
ET :=

//Para memorizar que se ha pulsado la activación (E0.0) se setea una marca.

U "Activacion"
S "Memoria_activación"

// Cuando se han cerrado las tolvas de azúcar, aditivos y concentrado, y se
// ha pulsado la activación (E0.0 memorizado en una marca) arranca el
// temporizador del agitador y se resetea la marca Memoria_activación.

UN "t1".Q
UN "t2".Q
UN "t3".Q
U "Memoria_activación"
= "Activacion_Agitador"
R "Memoria_activación"
```

2.º Ahora se crea un FB2 con los parámetros estáticos (STATIC) y del tipo FB1. Para hacer multiinstacias es obligatorio declarar estas variables STATIC. Las variables que se deben declarar son: néctar, zumo y refresco.

Para ello se debe abrir el FB2 y en la zona de parámetros STATIC incluir las variables y abrir el tipo buscando el tipo del FB1 (Figura 173).

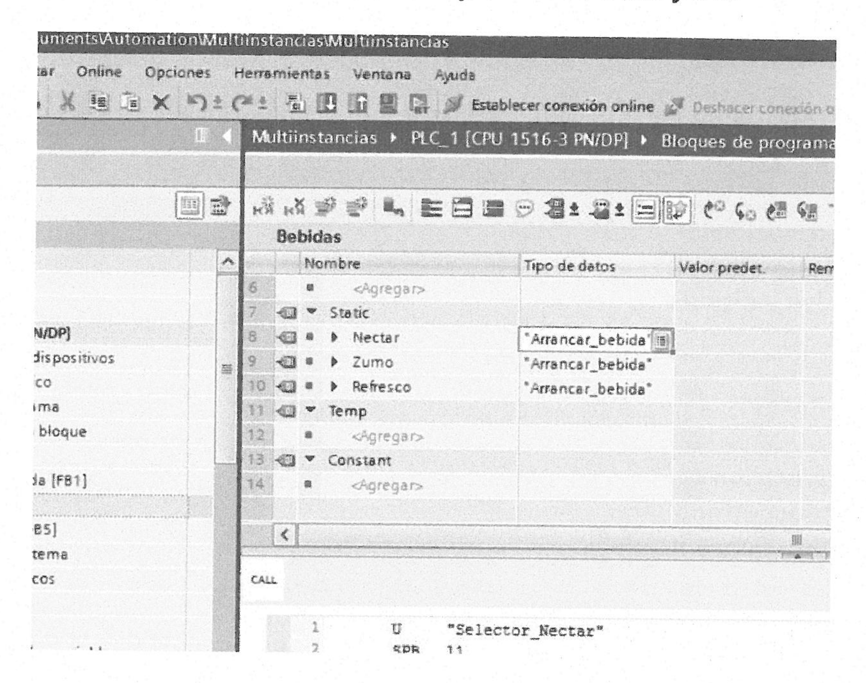


Figura 173

Ahora, y en este mismo FB2 (llamado Bebidas), se deben hacer las llamadas necesarias al FB1. Para ello se arrastra desde el árbol del proyecto el FB1 (Arranque_bebida) y al soltar sobre el FB2 obtenemos la pantalla de la Figura 174.

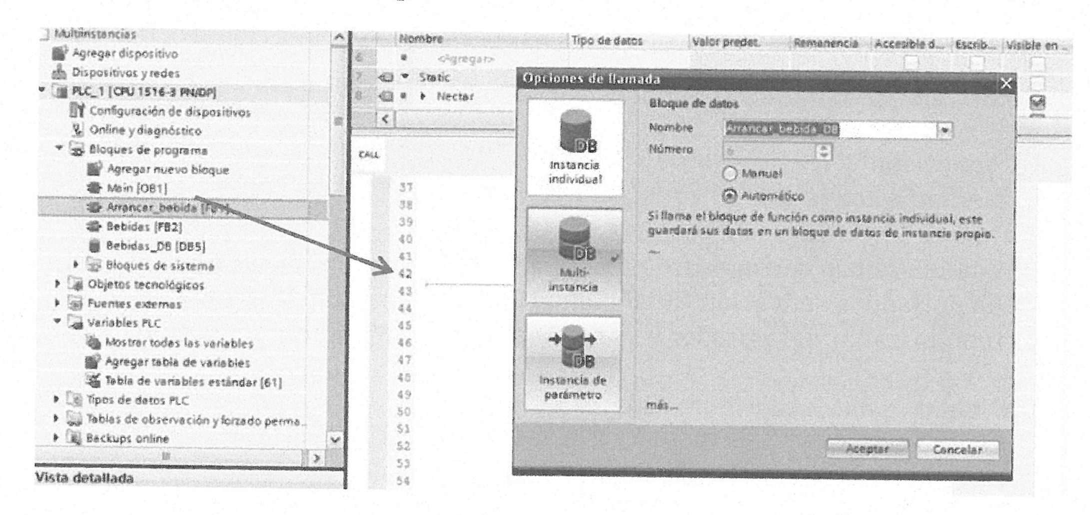


Figura 174

En esa pantalla se selecciona multiinstancia y otra pantalla como la de la Figura 175. Se debe seleccionar la instancia deseada entre las tres que figuran.

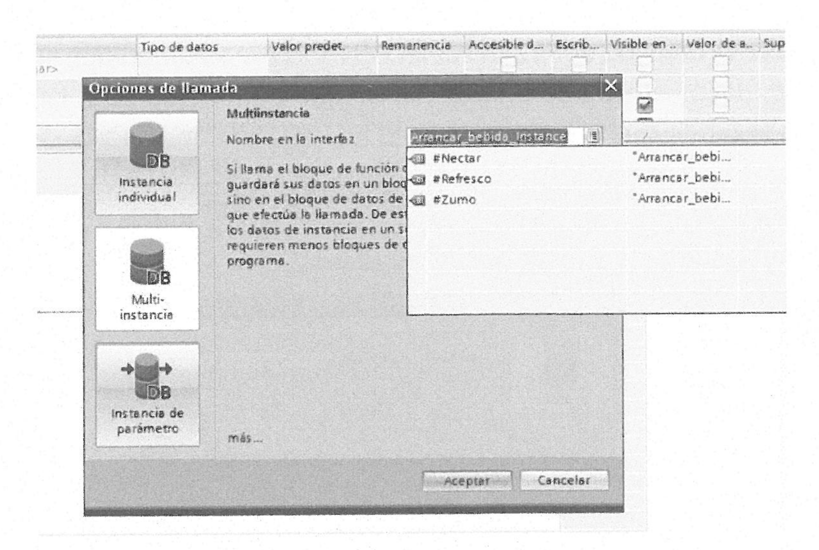


Figura 175

Una vez que se tienen las llamadas a las tres instancias, se crea el programa correspondiente que en este caso es la selección de una de las tres bebidas, para lo cual se dispone de tres interruptores/selectores.

El programa es el siguiente:

```
U    "Selector_Nectar"
SPB  l1

U    "Selector_Zumo"
SPB  L2

U    "Selector_Refresco"
SPB  L3
BEA

l1:  CALL #Nectar
       azucar      :=t#5s
       aditivos    :=T#6s
       concentrado :=t#4s
       agitador    :=t#6s
BEA

L2:  CALL #Zumo
       azucar      :=T#4s
       aditivos    :=t#2s
       concentrado :=t#10s
       agitador    :=t#4s
BEA

L3:  CALL #Refresco
       azucar      :=t#7s
       aditivos    :=t#8s
       concentrado :=t#2s
       agitador    :=t#10s
```

3.º Solo queda hacer la llamada al FB2 desde el OB1 y crear su DB al que se accede de forma múltiple. Para ello, desde el OB1 se arrastra el bloque de función FB2. La pantalla que nos aparece es la de la Figura 176.

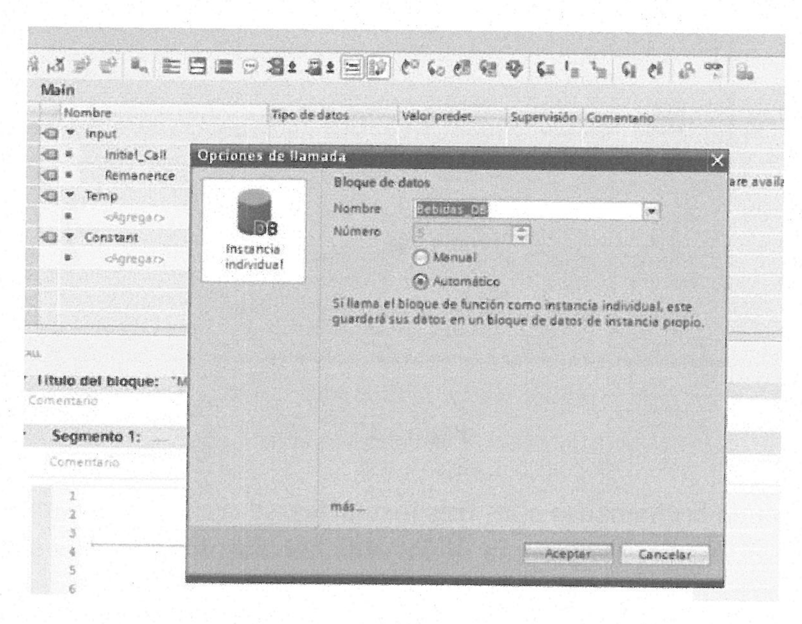


Figura 176

Se ha creado el DB y, si se abre, se aprecia que aparecen los datos de las tres bebidas (Figura 177). Con esto el programa está concluido.

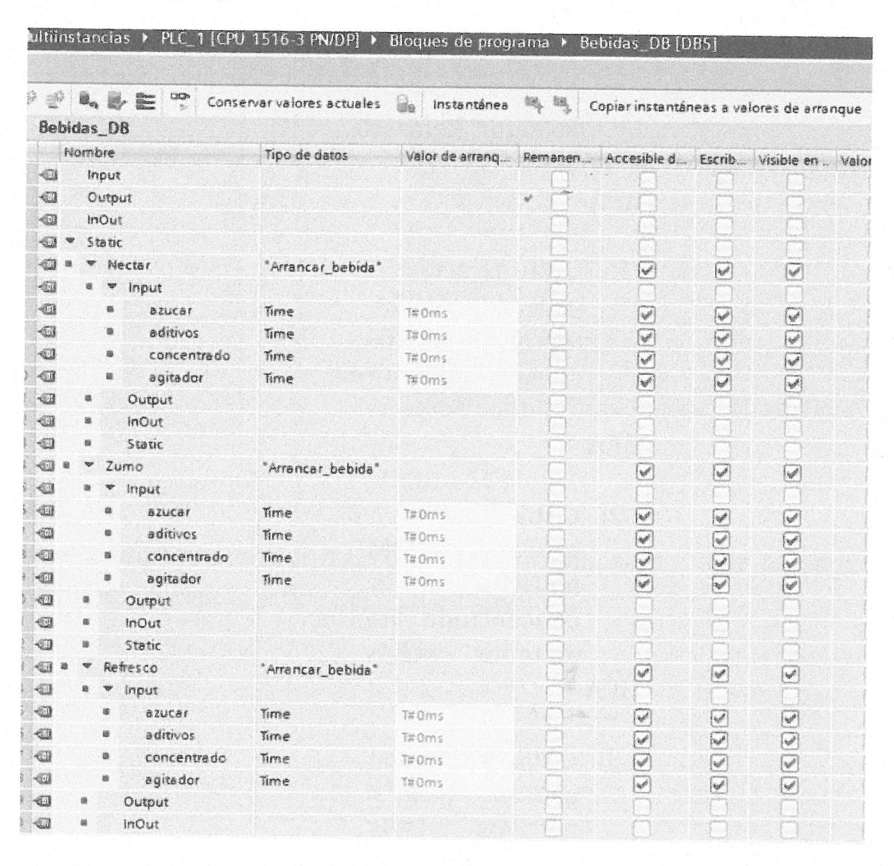


Figura 177

TIPO DE DATOS PLC (UDT) EN TIA PORTAL

En TIA PORTAL hay un tipo de datos que ya existía en las versiones anteriores de STEP 7. Este tipo de datos era el UDT, **Tipo de datos definidos por el usuario**. En TIA PORTAL ahora se denomina **Tipos de datos del PLC** y se encuentran en el árbol del programa.

Los **Tipos de dato del PLC** son estructuras de datos que el usuario puede definir a su conveniencia y pueden ser datos de tipos diferentes. Se pueden mezclar datos de tipo entero con datos de tipo tiempo o realizar cualquier otra combinación. Si se definen como tipos de datos del PLC, se pueden utilizar en un programa cuantas veces se desee y utilizar como plantillas de datos en los parámetros de un FB. Una ventaja a destacar es que, si se produce alguna variación, como puede ser incluir una nueva variable en esa estructura previamente definida, se actualizará en todos aquellos lugares donde se haya colocado. Con ello se evita el tener que realizar las modificaciones una a una. Otra de las ventajas de usar este tipo de datos es que se aprovecha al máximo el rendimiento del PLC S7-1500.

El procedimiento para crear estos datos es el siguiente:

1.º Desde el árbol del proyecto, en el apartado **Tipo de datos del PLC**, se agrega un nuevo tipo de datos y se cambia el nombre al tipo de datos creado por alguno relacionado con su utilización.

2.º Se abre la tabla de datos nueva haciendo doble clic sobre el nombre.

3.º Se introducen los diferentes datos y sus tipos.

Para aclarar y poder ver una aplicación se va a utilizar el mismo ejercicio empleado en las multiinstancias. En este caso no se van a utilizar multiinstancias, se crearán tres FB cada uno con su DB asociado. El FB empleará el tipo de datos de PLC definido para esta ocasión.

Al ejercicio anterior se le va a añadir una alarma por cada una de las tolvas, de forma que si está cerca de vaciarse una lámpara, avisará de la situación. Para ello dispone de un sensor que detectará si se llega a esa situación.

La tabla de variables utilizada es la misma que en el ejercicio anterior y que se puede ver en la Figura 171.

Se debe seguir el siguiente procedimiento:

1.º Crear la tabla de tipo de datos del PLC. La Figura 178 indica los datos creados para el ejercicio. Se le ha asignado el nombre de BEBIDAS.

2.º Se va a crear un FB por cada bebida y se colocarán como parámetro STATIC los parámetros locales creados en el tipo de datos del PLC. EL tipo de dato será BEBIDAS, según se puede ver en la Figura 179. Se puede crear un programa de una bebida y las demás FB se copian y pegan.

3.º Se realizan las llamadas a los FB correspondientes en función de la selección de bebida deseada.

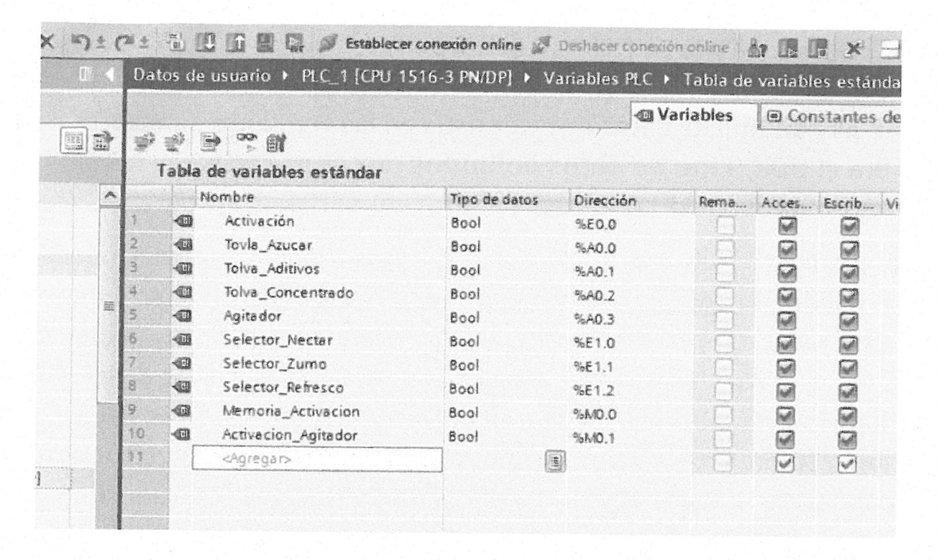


Figura 178

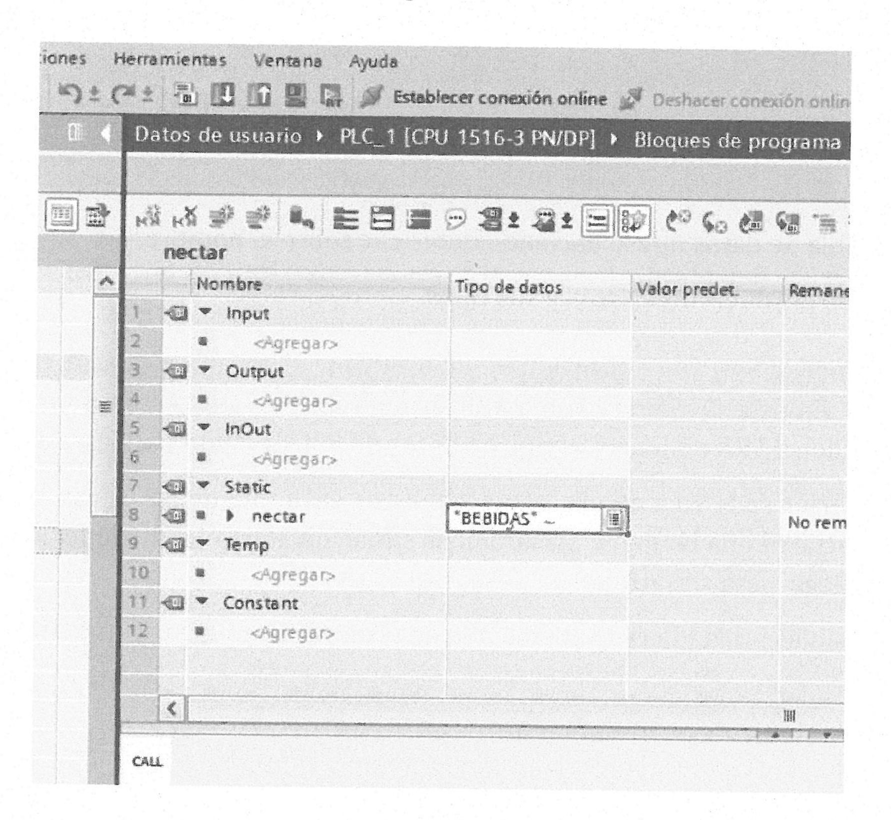


Figura 179

Según se observa, la variable néctar creada con el tipo de dato definido por el usuario como BEBIDAS tiene todas las variables creadas en los tipos de datos del PLC. Al desplegar el parámetro, se pueden ver las variables. Véase la Figura 180.

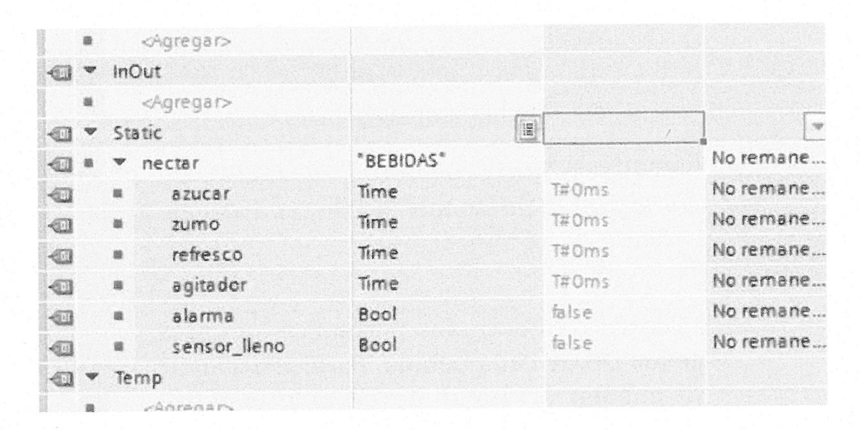


Figura 180

El programa del FB Néctar es el mismo que ya se había hecho en el ejercicio anterior. Los FB de zumo y refresco son idénticos, excepto por los tiempos que corresponden a cada bebida.

```
CALL  TP , "t1"
  time_type:=Time
  IN :="Activación"
  PT :=t#4s
  Q  :="Tovla_Azucar"
  ET :=

CALL  TP , "t2"
  time_type:=Time
  IN :="Activación"
  PT :=t#2s
  Q  :="Tolva_Aditivos"
  ET :=

CALL  TP , "t3"
  time_type:=Time
  IN :="Activación"
  PT :=t#10s
  Q  :="Tolva_Concentrado"
  ET :=

CALL  TP , "t4"
  time_type:=Time
  IN :="Activacion_Agitador"
  PT :=t#2s
  Q  :="Agitador"
  ET :=

U   "Activación"
S   "Memoria_Activacion"

    UN   "t1".Q
    UN   "IEC_Timer_0_DB".Q
    UN   "IEC_Timer_0_DB_1".Q
```

```
U   "Memoria_Activacion"
=   "Activacion_Agitador"
R   "Tolva_Aditivos"
```

En el OB1 se realizan las llamadas a las distintas bebidas en función de la selección deseada. Se añaden las entradas de los sensores de las tolvas y de cada alarma.

El programa es el siguiente:

```
// Estas tres entradas pertenecen a datos
de los DB de cada bebida. Para seleccionar
se pueden arrastrar desde el propio DB
hacia el OB1. Para ello se deberá dividir la
pantalla para que aparezca el OB1 y el DB
(Figura 181).

    U   "nectar_DB".nectar.sensor_vacio
    =   "nectar_DB".nectar.alarma

    U   "refresco_DB".refresco.sensor_vacio
    =   "refresco_DB".refresco.alarma

    U   "zumo_DB".zumo.sensor_vacio
    =   "zumo_DB".zumo.alarma

    U   "Selector_Nectar"
    SPB  l1

    U   "Selector_Zumo"
    SPB  l2

    U   "Selector_Refresco"
    SPB  l3
    BEA

l1: CALL "nectar", "nectar_DB"
    BEA

l2: CALL "refresco", "refresco_DB"
    BEA

l3: CALL "zumo", "zumo_DB"
```

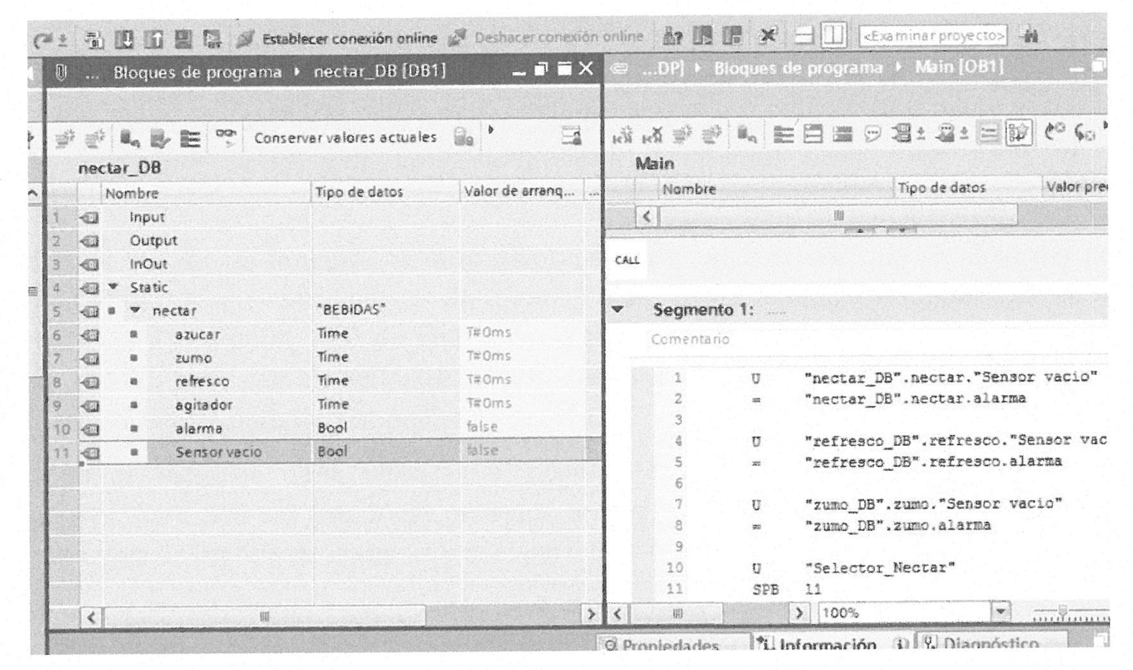


Figura 181

Como se ha podido observar al utilizar estructuras de tipo datos del PLC, se evita tener que declarar individualmente todos los parámetros de forma repetitiva. Y lo que es muy importante, si se introduce una nueva variable en este tipo de datos, quedará reflejado en todos los bloques donde se utilizan. Para comprobar este hecho, se puede abrir la tabla del tipo de datos y agregar una nueva línea, escribir una nueva variable y compilar. Se observará cómo en cualquiera de los FB ha aparecido dicha variable.

En la Figura 182 se puede observar la inclusión de la variable sensor lleno, cómo aparece ya en todos los DB y parámetros de los FB, y por lo tanto se podría utilizar.

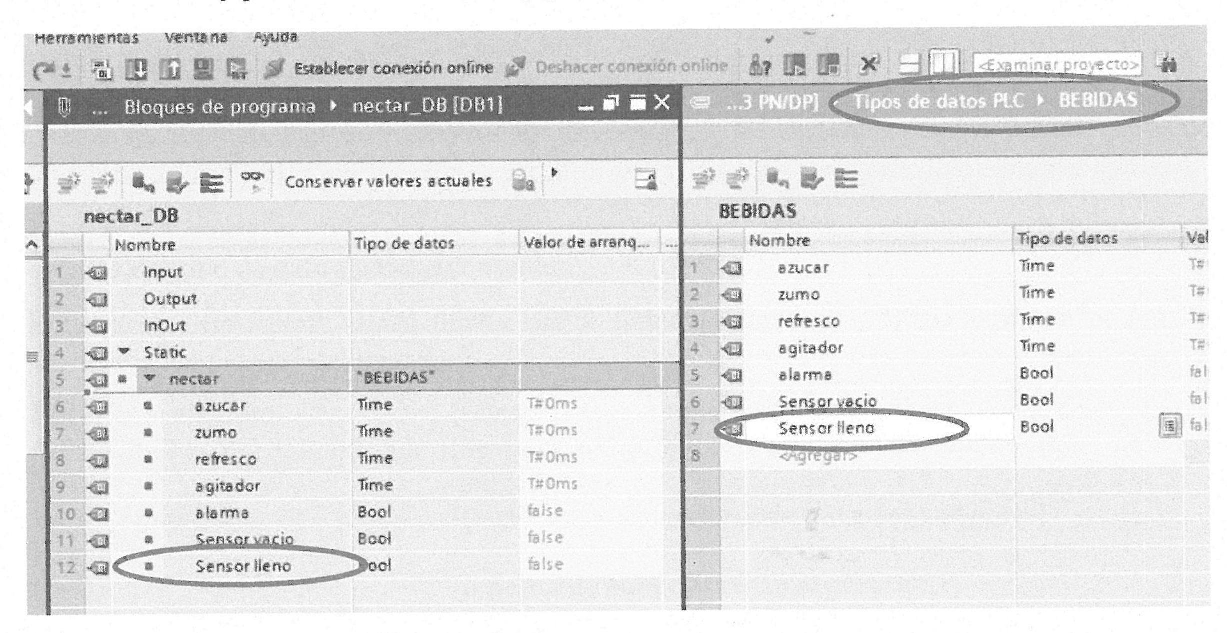


Figura 182

15. LENGUAJE SCL

INTRODUCCIÓN

El lenguaje SCL es un lenguaje de alto nivel, es decir, un lenguaje que se aleja del que la máquina entiende, que son códigos de unos y ceros. Es un lenguaje estándar que sirve, en principio, para cualquier PLC de cualquier fabricante. Utiliza instrucciones de texto, algo similar al AWL pero con un lenguaje de tipo más informático. Sigue la norma IEC 61131 ya vista anteriormente para los temporizadores.

Este tipo de lenguaje tiene ventajas e inconvenientes, así como grandes defensores y detractores. Es difícil ser imparcial en estos temas, ya que interviene el dominio que una persona tenga de un tipo de lenguaje y la experiencia en el mismo.

El intento de ser un lenguaje estándar lo hace muy atractivo y es una de sus grandes ventajas. De igual manera, hay que tener en cuenta que para poder programar en SCL es necesario tener destrezas y conocimientos de programación de lenguajes de alto nivel, tipo C o Pascal. Este hecho hace que aquellas personas que proceden del ámbito más industrial o técnico se inclinen más por los lenguajes de contactos, como el KOP.

Las aplicaciones de esta programación son aquellas que requieren unas características concretas, como por ejemplo programas donde hay muchas y muy complejas operaciones aritméticas como son aquellos que utilizan valores analógicos y se requiere procesar muchos datos; programas que utilizan muchas matrices, como ocurre en almacenes inteligentes donde se utiliza una alta gestión de datos y de recetas. Los ejemplos que se van a utilizar en este libro solo pretenden ser una iniciación a SCL, con ejemplos sencillos. Es evidente que ninguno de ellos precisa utilizar dicho lenguaje.

PRINCIPIOS BÁSICOS DE PASCAL

El lenguaje SCL está basado en Pascal. En este lenguaje las variables pueden utilizar datos de tipo: entero (integer), carácter (char), bit (bolean), real (real) y cadena de caracteres (string).

Es necesario declarar todas las variables que se van a utilizar, indicando el nombre de la variable y el tipo de dato que va a contener.

Un programa de Pascal básico puede ser el siguiente (las palabras en mayúsculas son las palabras clave propias del lenguaje):

```
PROGRAM ejemplo;                 {Con esto se pone nombre al programa}
CONST numero1= 212; n2=54;       {Se puede dar valor (constante ) a un nombre, no es
                                 obligatorio}
VAR   a:Integrer;                {Declaración obligatoria de todas las variables}
b:real;
BEGIN                            {A partir de aquí comienza el programa}
____                             {Aquí se colocan todas las instrucciones del programa}
____

END.                             {Con esto se termina el programa, incluido el punto}
```

Lo que está entre llaves son los comentarios. También se pueden introducir comentarios con: (*.......*) o con //.

Instrucciones básicas:

- Asignación := Con la asignación se pasan valores de una expresión a una variable o un valor a una variable. La expresión puede ser otra variable: *suma:=a+b+3;*
  a:=b;

Después de cada expresión o instrucción debe colocarse un punto y coma. En el lado derecho de una asignación solo puede haber una variable. El sentido de la información siempre va de derecha a izquierda, y no al revés. Esto no sería posible *a+b+3:= suma.*

- For

La sentencia *For* repite un número de veces la orden, o grupo de órdenes, que hay dentro de la sentencia.

La estructura es:

FOR *variable* **:=** *número inicio* **to** *número final* **DO**

Sentencia1;

Ejemplo:

FOR conta :=0 to 10 DO

a= a+1;

Donde la *Sentencia1* a=a+1; se repetirá 10 veces.

Puede haber más de una sentencia dentro del lazo si se coloca de este modo:

FOR *variable* **:=** *número inicio* **to** *número final* **DO**

BEGIN

Sentencia1;

Sentencia2;

END;

La instrucción BEGIN abre un grupo de sentencias que se deben cerrar con END;

- While

La sentencia While repite una sentencia o grupo de sentencias mientras se cumpla la condición que lleva asociada la orden While.

La estructura de While es la siguiente:

WHILE *condición* **DO**

Sentencia1;

Para ejecutar más de una sentencia:

WHILE *condición* **DO**

BEGIN

Sentencia1;

Sentencia2;

END;

Ejemplo:

```
WHILE a>8 DO
  BEGIN
  a:=a+1;
  b:=a+2;
  END;
```

- If.....Then

La instrucción IF....THEN ejecuta la sentencia o grupo de sentencias que lleva la orden solo si se cumple la condición del IF. Si es falsa, no se ejecutan y prosigue el programa con las órdenes que están a continuación del IF. Para incluir varias sentencias, se deben poner entre BEGIN y END;

Estructura de la instrucción IF THEN:

IF condición **THEN**

sentencia1;

sentencia2;

En este caso, si se cumple la condición, se ejecuta la *sentencia1* y luego continúa con *sentencia2*. Si no se cumple la condición, no se ejecuta la *sentencia1* y pasa directamente a la *sentencia2*.

- If.....Then Else

Esta es una variación de la anterior. Si se cumple la condición, se ejecuta la sentencia o grupo de sentencias que lleve Then. Si no se cumple, se ejecuta la sentencia o grupo de sentencias que lleva Else.

Estructura de la instrucción IF THEN ELSE:

IF condición **THEN**

sentencia1;

ELSE sentencia2;

sentencia3;

Si se cumple la condición, se ejecutará la *sentencia1* y el programa seguirá con *sentencia3*. Si no se cumple la condición, se ejecutará *sentencia2* y a continuación el programa seguirá con *sentencia3*.

- If, Elsif y Else

Esta instrucción es una variante de la anterior y en este caso lleva dos condiciones. La estructura de esta orden es la siguiente:

IF condición1 **THEN**

sentencia1;

ELSIF condición2

sentencia2;

ELSE sentencia3;

Sentencia4;

Si se cumple la condición1, se ejecuta la sentencia1 y el programa sigue con sentencia4. Si no se cumple la condición1, se evalúa la condición2. Si es verdadera, se cumple sentencia2 y sigue con sentencia4. Si no se cumple ni la condición1 ni la condición2, se ejecuta ELSE, es decir, la sentencia3.

Ejemplo: *IF a<10 THEN*

a:=a+1;

ELSIF b>15

a:= a+2;

ELSE a:=a+3;

b:=0;

Si la variable **a** es menor que 10, se le sumará 1 a la variable **a** y luego continuará dejando un cero en la variable **b**. Si **a** no es menor que 10, se comprueba si **b** es mayor que 15 y, si es cierto, suma dos a la variable **a**. A continuación, sigue poniendo un cero en la variable **b**. Pero si tampoco esta condición es cierta, se ejecuta la sentencia, o grupo de sentencias, que lleva ELSE. En este caso añade 3 a la variable **a** y el programa seguirá poniendo un cero en la variable **b**.

- Case

La instrucción CASE se utiliza para seleccionar entre varias opciones.

Su estructura es:

```
CASE expresión OF
        1: sentencia1;
        2: sentencia2;
 3: sentencia3;
ELSE sentencia4;
sentencia5;
```

La expresión será una variable numérica o de tipo carácter. Si el valor de la variable coincide con algún número de los que están colocados con los dos puntos (:), ejecutará la sentencia que lleva dicho número. Después de ejecutarla, continuará con la sentencia5. Si el valor de la variable contiene un valor que no existe en CASE, se ejecutará la sentencia de ELSE, sentencia4, siguiendo después con la sentencia5. La instrucción ELSE es opcional. Si no se incluye, en el caso de no existir el número en CASE, pasaría a ejecutar la sentencia5.

Ejemplo:

```
CASE a OF
        1:c:=c+1;
        2:c:=c+2;
        3:c:=c+3;
ELSE c:= 0;
c:=c+4;
```

Si el valor de la variable **a** es uno, se sumará uno a la variable **c**. Si el valor de **a** es 2, se le sumará 2 a **c** y si **a** es igual a 3, se suma 3 a la variable **c**. Si el valor de **a** no es ni 1 ni 2 ni 3, se ejecuta el ELSE, es decir, se pone a cero la variable **c**. Después de cualquier ejecución, el programa continuará sumando 4 a la variable **c**.

Si se desea ejecutar más de una sentencia en cada una de las selecciones del CASE, es suficiente ponerlas entre BEGIN y END;

- Repeat.....Until

Esta instrucción es similar al While. En esta se repite la sentencia hasta que se cumpla la condición que lleva Until. Si en vez de una sentencia se desea disponer de más de una, se colocarán entre BEGIN y END. Hay una diferencia importante entre el Repeat y el While. En Repeat se van a ejecutar las ordenes siempre al menos una vez, ya que primero ejecuta y al final evalúa la condición. En el While la evaluación de la condición lo hace al principio.

La estructura es la siguiente:

REPEAT
Sentencia1;
UNTIL condición;

sentencia2;

En este caso repite la sentencia1 hasta que se cumpla la condición de UNTIL. Una vez terminado el ciclo, el programa continúa con la sentencia2.

- Continue

Mediante esta orden se consigue **salir de la situación actual de un bucle**. Después de ejecutarse, sigue el bucle normalmente y solo deja de ejecutarse la acción en ese momento del bucle. Se puede utilizar en las instrucciones While, Repeat y For.

- Exit

Es una orden similar a la anterior, pero en este caso sale inmediata y permanentemente del bucle. Su comportamiento es el mismo que si se terminara el bucle de forma natural. Solo se puede utilizar en las instrucciones While, Repeat y For

- Goto

Esta sentencia modifica la ejecución secuencial normal del programa llevándolo a otra secuencia del mismo. Es una orden que no se debe utilizar en los lenguajes de tipo estructurado porque consigue justo todo lo contrario: romper y desordenar el programa.

Expresiones aritméticas y lógicas:

Operador	Operación	Jerarquía
Expresiones aritméticas		
+	Más unario	2
-	Menos unario	2
**	Potencia	3
*	Multiplicación	4
/	División	4
MOD	Función módulo	4
+	Suma	5
-	Resta	5

Expresiones de comparación		
<	Menor	6
>	Mayor	6
<=	Menor o igual	6
>=	Mayor o igual	6
=	Igual	7
<>	Diferente	7
Expresiones lógicas		
NOT	Negación	3
AND o &	Y booleano	8
XOR	O-exclusiva	9
OR	O booleano	10
Otras operaciones		
()	Paréntesis	1
:=	Asignación	11

- Arrays y matrices

Las matrices son agrupaciones de datos de un mismo tipo; también se denominan «arreglos». Pueden ser de varias dimensiones: una dos, tres, etc. Los *arrays* son matrices de una sola dimensión y también se denominan vectores.

Se caracterizan por disponer de muchos datos, todos guardados de forma contigua en la memoria. Cada posición de memoria de una matriz dispone de un mismo identificador y un índice. A cada posición de memoria se le denomina elemento y se accede a cada elemento mediante el índice. Es muy importante tener en cuenta que todos los elementos de una matriz son del mismo tipo. No se pueden mezclarse elementos de una misma matriz que contenga datos enteros y reales.

Una matriz unidimensional llamada M1 de 5 elementos estará formada por 5 posiciones de memoria y cada una se denominará de la siguiente forma:

M1[0], M1[1], M1[2], M1[3], M1[4]. El primer elemento de una matriz siempre es el cero.

Modificando el valor del índice se accede a los diferentes elementos de una matriz. Para acceder a un elemento de una matriz se debe hacer mediante su identificador y el índice del elemento con el que se desea trabajar. Por ejemplo, con M1[4] se hace referencia al elemento 4 de la matriz M1.

Siempre que se desee operar entre varias matrices hay que hacerlo elemento a elemento. Es decir, si se dispone de dos matrices M1 y M2, de cinco elementos y se desea multiplicarlas, no será válido hacer: M3:=M1 * M2, donde M3 es otra matriz de cinco elementos. Hay que ir operando elemento a elemento y para ello se utilizará la iteración FOR.

Previamente, como cualquier variable, hay que declarar las matrices indicando su nombre (identificador), las dimensiones, la longitud máxima de cada dimensión y el tipo de datos que va a contener. Una matriz de una dimensión se declararía de este modo:

Numero:ARRAY[1..20] OF integer; Es una matriz unidimensional denominada «Numero» que incluye 20 datos (del elemento 0 al elemento 19) y que contendrá números enteros. Estos elementos son:

Numero[0] Numero[1] Numero[2].......... Numero[19]

Numero:ARRAY[1..2, 1..5] OF real; Es una matriz de dos dimensiones denominada «Numero» que incluye 2 columnas y 5 filas, y que contendrá números reales. En total tendrá 10 elementos y estos serán:

Numero[0][0]Numero[1][0]
Numero[0][1]Numero[1][1]
Numero[0][2]Numero[1][2]
Numero[0][3]Numero[1][3]
Numero[0][4]Numero[1][4]

- Funciones y procedimientos

Las funciones y los procedimientos son módulos de programación que se ejecutan cuando son llamados desde un bloque de programación. Una vez ejecutada la función o el procedimiento, se vuelve al bloque del programa desde el que ha sido llamado.

Se les puede pasar parámetros o no. La función puede devolver un valor o no. Cuando a una función se le pasan parámetros, habrá que definirlos dentro de la función. Estos parámetros definidos dentro de la función se denominan «parámetros formales». Cuando se llama a la función hay que pasar los valores de dichos parámetros y a estos parámetros que se le pasa a la función se denominan «parámetros actuales». Si una función o procedimiento no tiene parámetros formales declarados, es evidente que tampoco tendrá parámetros actuales. Hay que definir previamente el tipo de dato de esos parámetros, es decir, si son booleanos, enteros, reales, etc.

La diferencia entre el procedimiento y la función es que el procedimiento no puede devolver datos y la función sí. Cuando se devuelve un dato, la función se comporta como una variable con la que puede operarse y cuyo contenido es el valor devuelto desde la función. Hay que definir el tipo de dato que se va a devolver, es decir, si el dato que se devuelve es booleano, entero, real, etc. Si no se va a devolver ningún valor, habrá que declararlo como «void».

Como el uso de las funciones en SCL es algo diferente al Pascal, se evita dar más explicaciones en este momento. Se estudiará la forma de utilizar las funciones en el apartado correspondiente de SCL. En SCL solo existen funciones.

SCL EN TIA PORTAL V13 PARA PLC S7-1500

En los apartados anteriores se ha descrito el lenguaje Pascal de forma básica. El uso de SCL está basado en Pascal, pero las cosas son algo diferentes a las tratadas anteriormente y, en los entornos de Siemens, ha ido evolucionando de forma que su programación es más sencilla. Por ejemplo, en la última versión, en TIA PORTAL (V13) para PLC S7 1500 no hay que poner el título del programa, ni el comienzo ni final del programa, ni hay que declarar las variables. Las variables se declaran en la tabla de variables del PLC si van a ser globales, o en la interfaz de usuario de la propia función si van a ser locales. Son variables globales aquellas que se pueden usar en cualquier bloque (OB1, FB, FC...) y variables locales aquellas que solo son válidas en el propio bloque en el que se declaran.

En los siguientes apartados se va a centrar el estudio del lenguaje SCL en el entorno TIA PORTAL (V13) y para los PLC S7 1500. Los ejemplos van a ser sencillos y tendrán como único objetivo la perfecta comprensión de cada una de las sentencias SCL y su aplicación directa al PLC.

La forma adecuada de trabajar es utilizar diversas funciones (FC o FB). Estas funciones pueden estar escritas en diferentes lenguajes dependiendo de sus necesidades. El OB1 realiza las llamadas a las diferentes funciones.

Si se va a trabajar en SCL, lo primero que se debe hacer es declarar todas las variables que se van a utilizar en el programa de SCL. Se hace desde la tabla de variables. Al ser declaradas desde la tabla de variables, todas serán globales, es decir, se podrán utilizar en cualquier bloque que se desee.

Se van a declarar las siguientes variables (en la tabla de variables):

Entradas:
Entrada0... E0.0, Entrada1... E0.1, Entrada2... E0.2, Entrada3... E0.3, Entrada4... E0.4, Entrada5... E0.5, Entrada6... E0.6, Entrada7... E0.7

Salidas:
Salida0... A0.0, Motor1... A0.1, Motor2... A0.2, Motor3... A0.3, Salida4... A0.4, Salida5...A0.5, Salida6...A0.6, Salida7...A0.7.

Para poder programar en SCL hay que agregar un nuevo bloque y seleccionar el lenguaje de programación, tal como se ve en la Figura 183.

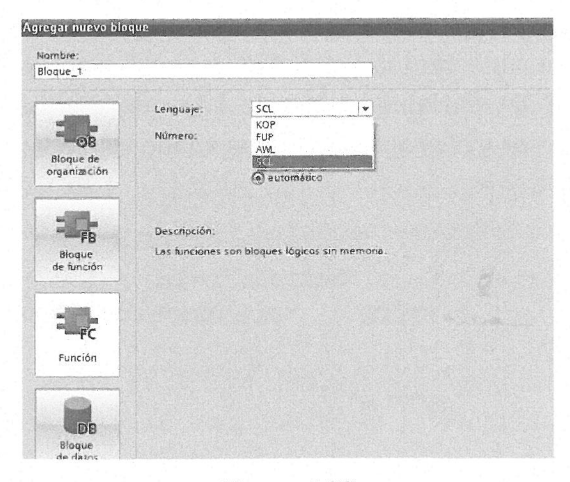


Figura 183

En la pantalla del bloque que se ha abierto para programar en SCL se pueden observar diferentes apartados. Por una parte se aprecia el editor para escribir el programa y encima un grupo de **sentencias de acceso rápido**. Encima del editor se encuentra la definición de los parámetros, que en este caso serán también las **variables locales**, es decir, solo serán válidas dentro del bloque. Y a la derecha se encuentran todas las **instrucciones** que se pueden utilizar en SCL. En la **parte inferior** se puede observar el resultado de la compilación. Esto se aprecia en la Figura 184.

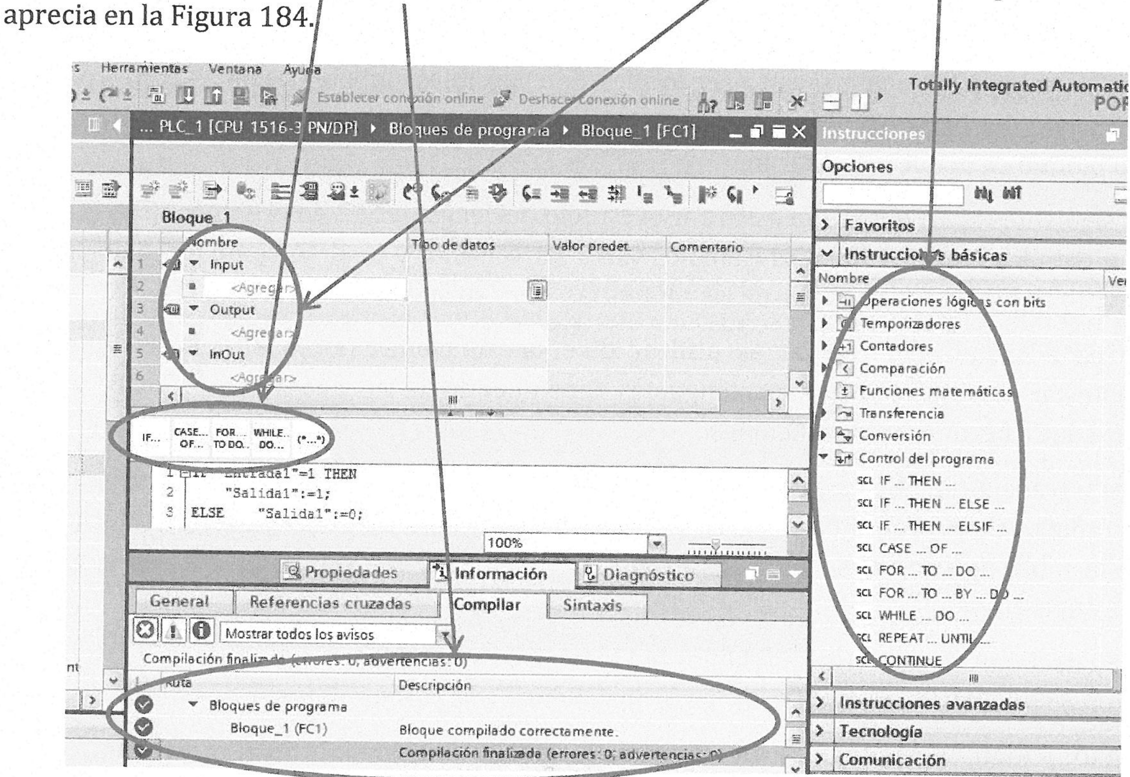


Figura 184

Ejemplos sencillos en SCL

Puesta en marcha de un motor mientras se acciona un pulsador

Cuando se coloca la orden IF, se pone de forma automática la estructura necesaria sin que el programador deba preocuparse por eso. En el ejemplo se observa el programa, de forma que, si se activa la Entrada1, se conectará la Salida1. Si no está activada la Entrada1, la salida se pone a cero. Para colocar las variables, tan solo hay que escribir alguna de las variables declaradas en la tabla anterior y el propio programa añade las dobles comillas.

```

IF "Entrada1"=1 THEN
    "Salida1":=1;
ELSE    "Salida1":=0;

END_IF;

```

Se debe observar la diferencia del igual (= , :=) en la sentencia condicional del IF y en la expresión. Siempre que se introduce una expresión condicional de igualdad, se pone solo el igual (=). Sin embargo, cuando lo que se desea es expresar que una variable debe contener un valor, siempre se debe anteponer al igual los dos puntos (:=).

Paro/Marcha de un motor

En este caso se desea arrancar un motor mediante un paro/marcha clásico. Para ello se necesitarán dos pulsadores: uno para parar y otro para poner en marcha. El pulsador de paro debe ser un pulsador normalmente cerrado y el de marcha, normalmente abierto.

```
IF "Entrada1" = 1 THEN
    "Salida1" := 1;

END_IF;
IF "Entrada2"=0 THEN
    "Salida1" := 0;

END_IF;

```

Si la Entrada1, que es la marcha, no está activada, el motor no arranca. Cuando se activa el pulsador de marcha, la Salida1 se activa. Si la Entrada2, que es el paro y está normalmente cerrado, se activa (es cero), entonces se pone a cero la Salida1.

Un motor condicionado al arranque de otro

Con un pulsador se arranca el Motor1. Con otro pulsador arranca el Motor2 solo si ha arrancado el Motor1. Con otro pulsador se paran los motores que estén en marcha.

```
IF "Entrada1" THEN
    "Motor1":=1;
END_IF;
IF "Motor1" AND "Entrada2" THEN
    "Motor2":=1;
END_IF;
IF "Entrada3"=0 THEN
    "Motor1" := 0;
    "Motor2" := 0;
END_IF;
```

Con la Entrada1 arranca el Motor1. Para que arranque el Motor2 es necesario que haya arrancado el Motor1 y que se accione la Entrada2. Se debe observar el operador lógico AND (también se puede escribir &) que une la doble condición para que arranque el Motor2. Accionando la Entrada3 (está normalmente cerrado) se paran los dos motores.

Ejemplos con temporizadores IEC en SCL

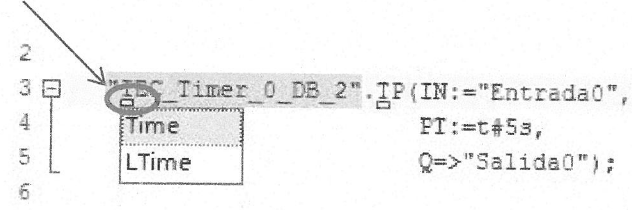
Trabajar con temporizadores IEC en SCL es similar a como se hace en AWL. Se selecciona el tipo de temporizador deseado y se arrastra desde la ventana de instrucciones básicas. Aparece algo similar a cuando se utilizaba en el lenguaje AWL. Antes nos pedirá el nombre del DB que se va a utilizar. Este ejemplo es para un temporizador TP:

```
"IEC_Timer_0_DB_1".TP(IN:=_bool_in_,
                      PT:=_in_,
                      Q=>_bool_out_,
                      ET=>_out_);
```

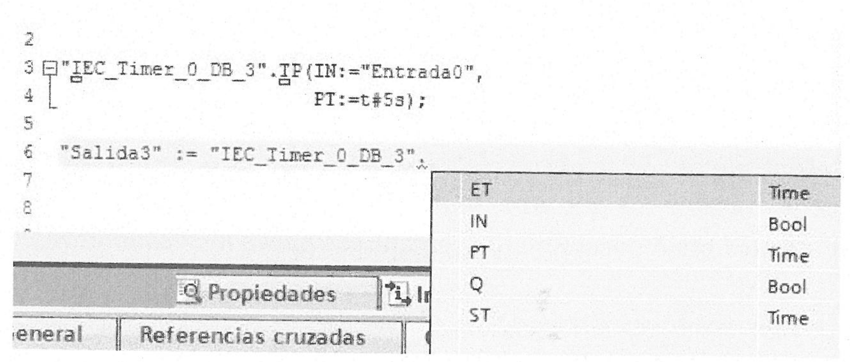
Se coloca el valor de la entrada que arrancará el temporizador (IN), el valor del tiempo a temporizar (PT), la salida sobre la que actúa (Q) y la variable donde se va a dejar el tiempo actualizado (ET).

Para colocar los diferentes valores hay que tocar dos veces, con el botón izquierdo del ratón, sobre el lugar donde se va a colocar el dato.

Si no se desea dar algún valor, se deja sin tocar, tal como aparece cuando se ha colocado el temporizador. Al pulsar ENTER, se eliminará esa opción. Además también aparecen unos pequeños rectángulos de color verde. Pulsando sobre ellos se podrá cambiar el tipo de datos del temporizador (Time o LTime) y el tipo de temporizador. En el ejemplo siguiente se pueden ver los **rectángulos verdes.**



Se podrá hacer referencia a cualquier parámetro del temporizador nombrándolo en cualquier parte del programa. Si no se coloca la variable en Q al parametrizar el temporizador, se puede asignar la salida del temporizador a cualquier variable. Para ello se empieza escribiendo el nombre del temporizador: IEC... Se mostrará una ventanita con los temporizadores disponibles y se escoge el deseado. A continuación, se pone un punto y aparecen todos los parámetros del temporizador. Se selecciona Q en este caso. No hay que olvidarse de añadir el punto y coma (;) de fin de orden después de escribir Q. En el ejemplo siguiente se muestra un ejemplo.



Quedaría así:

```

"IEC_Timer_0_DB_3".TP(IN:="Entrada0",
                      PT:=t#5s);

"Salida3" := "IEC_Timer_0_DB_3".Q;

```

En el siguiente ejemplo se arranca un temporizador con una entrada ("Entrada0") solo si antes está el acceso permitido con otra entrada ("Entrada1").

```
IF "Entrada1" THEN
    "IEC_Timer_0_DB".TP(IN:="Entrada0",
                        PT:=t#5s,
                        Q=>"Salida0");
END_IF;

```

Ejemplos con contadores IEC en SCL

Con los contadores IEC pasa lo mismo que con los temporizadores, se trabaja de forma similar a como se hace en AWL.

Incluiremos varios ejemplos donde se puede comprobar la forma de utilizarlos.

En el siguiente ejemplo se coloca un contador de subida, otro de bajada, y otro de subida y bajada. Si hay parámetros que no aparecen es porque no se han utilizado, como ocurría con los temporizadores.

```

"IEC_Counter_0_DB".CTU(CU:="Entrada0",
                       R:="Entrada1",
                       PV:=5,
                       Q=>"Salida0");

"IEC_Counter_0_DB_3".CTD(CD:="Entrada2",
                         LD:="Entrada3",
                         PV:=5,
                         Q=>"Salida3");


"IEC_Counter_0_DB_2".CTUD(CU:="Entrada4",
                          CD:="Entrada5",
                          R:="Entrada6",
                          LD:="Entrada7",
                          PV:=5,
                          QU=>"Salida4",
                          QD=>"Salida5");

```

Matrices en SCL

Trabajar con matrices en SCL no difiere mucho de hacerlo en cualquier otro lenguaje de alto nivel como C o Pascal.

Vamos a realizar un ejercicio en el que se utilizarán matrices dentro de un bloque de función (FB) y así se podrán observar, de una forma cómoda, los diferentes elementos de la matriz.

Ejercicio de matrices 1

El ejercicio trata de introducir datos en una matriz desde un byte de entradas del PLC.

Mediante el byte de entradas EB1 se introducirá el dato. Se va a utilizar una matriz unidimensional de 8 elementos. El número del elemento de la matriz se seleccionará con tres interruptores: E0.0, E0.1 y E0.2. Además, con la entrada E0.3 se activará la entrada del dato de EB1 hacia la matriz en su dirección seleccionada.

El procedimiento será el siguiente:

1.- Se selecciona el dato que se desea introducir en la matriz mediante EB1.

2.- Se selecciona el número de elemento donde se va a guardar el dato, mediante E0.0, E0.1 y E0.2.

3.- Se valida el dato en la matriz mediante E0.3.

Como variables del PLC se deberán declarar las siguientes:

La entrada de dato (EB1) que le denomina en este ejemplo dato.

La entrada de validación en la matriz, entrada (E0.3).

El índice, que estará formado por el byte de entrada EB0, pero siendo significativos tan solo los bits de menor peso: E0.0, E0.1 y E0.2. Se le llama Indice.

El programa se escribirá en FB1, creándose de forma automática un DB de instancia en el que se podrán ver (online) los elementos de la matriz. Se llamará al FB desde el OB1 de forma incondicional.

Se declaran como parámetros de entrada del FB (también podría declararse como algún tipo de dato local) la matriz y una variable que hará de índice de la matriz. La matriz, matriz1, se declara como array de 8 elementos de tipo entero y la variable indice1 también de tipo entero.

La entrada EB0 se utiliza como selector del índice de la matriz (los tres bits menos significativos) y también se utiliza el bit E0.3 como validador del dato de EB1 a la matriz. Se deberá realizar un filtro de EB0 para que pueda servir como índice. Para ello, todos los bits que no se utilizan como índice se ponen a cero (del E0.3 al E0.7). Para conseguirlo, se hace una operación lógica AND entre el byte de entrada EB0, que es la variable indice, y el entero 7 (0000 0111 B). Con esto todos los bits serán cero excepto los tres de menor peso que son los que hacen de índice.

#indice1 := "Indice" AND 7; // Filtro de la entrada EB0.

IF "entrada" = 1 THEN // Cuando se pulse la entrada E0.3

#matriz1[#indice1] := "dato"; // se introduce el dato de EB1 en elemento de la matriz indicado por "indice1" mediante E0.0, E0.1 y E0.3

END_IF;

En la Figura 185 se detalla el FB1 con sus parámetros definidos y en la Figura 186 la tabla de variables del PLC:

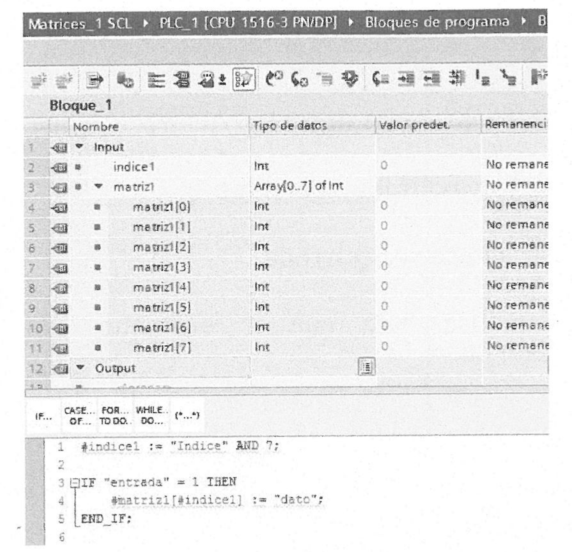


Figura 185

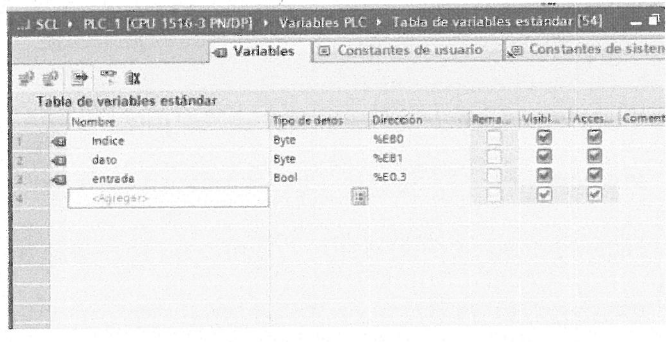


Figura 186

En el OB1 simplemente se realiza la llamada al FB, momento en el que solicitará crear un DB de instancia. Se acepta y se crea el DB. Todos los parámetros del FB se crearán en el DB. Al haber creado los parámetros como entrada, aparecerán en el OB1 al llamar al FB. Se dejan en blanco en este caso. De hecho, en SCL si no se dan parámetros, los quita, y esto sucede al accionar el ENTER cuando solicita los parámetros:

Al arrastrar el FB al OB1, nos muestra: *"Bloque_1_DB_3"(_int_inout_);*

Al pulsar ENTER, queda definitivamente así: *"Bloque_1_DB_3"();*

El número del DB que sale puede ser diferente al del ejercicio. Para ver el resultado, hay que observar la matriz en el DB poniendo las gafitas.

Ejercicio de matrices 2

Se va a modificar este ejercicio, de forma que se cree otra matriz partiendo de la anterior. La nueva matriz será la matriz original sumándole uno a cada uno de sus elementos. La operación se realizará al accionar el pulsador E0.4.

Al ejercicio anterior hay que añadir una nueva matriz con la misma dimensión y el mismo número de elementos, que se llamará *matriz2* y que se declarará en el FB como parámetro de entrada. También hay que declarar una variable del PLC que será la entrada con la que se acepta la suma de uno a la matriz original y crea la matriz nueva. Esta entrada es la E0.4 y se denomina *Sumar_1*.

Para hacer este ejercicio se debe recordar que no es posible operar con toda la matriz de una vez. Hay que realizarlo elemento a elemento.

El programa del FB es el siguiente:

#indice1 := "Indice" AND 7;

IF "entrada" = 1 THEN

#matriz1[#indice1] := "dato";

END_IF;

IF "Sumar_1"=1 THEN // La entrada E0.4 da la orden de sumar uno a la //matriz original

#indice1 := 0; // Se inicializa el índice a cero para rastrear todos los elementos

FOR #indice1:= 0 TO 7 DO //Como no se puede operar de una vez con toda

#matriz2[#indice1] := #matriz1[#indice1] + 1; //la matriz, hay que ir elemento //a elemento utilizando un lazo //for.

END_FOR;

END_IF;

Funciones en SCL

La forma de utilizar las funciones en SCL se ha mejorado en las últimas versiones de TIA PORTAL. Ahora es mucho más intuitivo trabajar con las ellas. Las funciones son subprogramas que se ejecutan en un bloque separado del programa principal. Se llaman desde el programa principal u otro bloque, abandonando este. Una vez que la función se ha ejecutado, se retorna de forma automática al bloque del programa desde el que se ha llamado, siguiendo la ejecución desde el punto donde se había dejado.

Hay que recordar que en los sistemas de programación de S7 de Siemens se tienen dos formatos de funciones: las funciones (FC) y los bloques de función (FB). La diferencia es que en el bloque de función se genera un bloque de datos (DB) vinculado al bloque de función.

A las funciones se les puede pasar parámetros (datos) en el momento de la llamada; estos parámetros se denominan «parámetros actuales». Para poder pasar parámetros es necesario crear las funciones (FC o FB) con dichos parámetros y los parámetros creados dentro de la función se denominan «parámetros formales». Además, una función puede actuar como una variable, es decir, puede tener un valor y operar con dicho dato como si fuera una variable. Así es como una variable puede retornar un valor.

Los parámetros formales se deben declarar en la zona de declaración de parámetros de cada bloque, llamada interface de bloque. Hay parámetros de bloque y parámetros locales. Los locales solo se utilizan en el bloque donde se han declarado y por ello no se pueden pasar valores desde el bloque que llama a estas funciones. Los parámetros de bloque sí pueden recibir datos del bloque que realiza la llamada, aunque tampoco es obligatorio hacerlo.

Los parámetros de bloque son:

Tipo	Sección	Función	Disponible en
Parámetros de entrada	Input	Parámetros cuyos valores lee el bloque.	Funciones, bloques de función y algunos tipos de bloques de organización
Parámetros de salida	Output	Parámetros cuyos valores escribe el bloque.	Funciones y bloques de función
Parámetros de entrada/salida	InOut	El bloque lee los valores de estos parámetros al efectuar la llamada y los vuelve a escribir en ellos tras la ejecución.	Funciones y bloques de función
Valor de retorno	Return	Valor que se devuelve al bloque que realiza la llamada.	Funciones

Los parámetros locales son:

Tipo	Sección	Función	Disponible en
Datos locales temporales	Temp	Variables que sirven para almacenar resultados intermedios temporales. Los datos temporales se conservan solo durante un ciclo. Si utiliza datos locales temporales, se debe asegurar que los valores se escriben (inicializan) dentro del ciclo en el que desea leerlos. De lo contrario, los valores serán aleatorios.	Funciones, bloques de función y bloques de organización **Nota:** En los bloques de datos de instancia no se visualizan los datos locales temporales.
Datos locales estáticos	Static	Variables que sirven para almacenar resultados intermedios estáticos en el bloque de datos de instancia. Los datos estáticos se conservan hasta que se vuelven a escribir, también a lo largo de varios ciclos.	Bloques de función
Constante	Constant	Constantes con nombres simbólicos declarados que se utilizan dentro del bloque. No son variables por lo que no se podrá enviar nada a ellos ya que son constantes.	Funciones, bloques de función y bloques de organización **Nota:** En los bloques de datos de instancia no se visualizan las constantes locales.

En la Figura 187 se puede observar el formato de la interfaz de bloque de un FB.

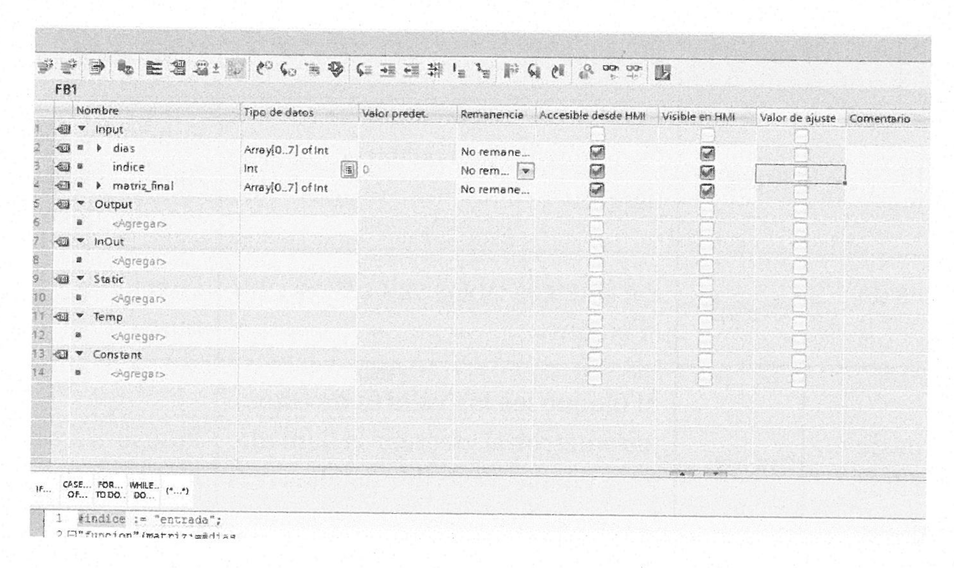


Figura 187

Columna	Significado
	Símbolo en el que se puede hacer clic para arrastrar un elemento mediante *Drag & Drop* hasta un programa y utilizarlo allí como operando.
Nombre	Nombre del elemento.
Tipo de datos	Tipo de datos del elemento.
Offset	Dirección relativa de una variable. Esta columna solo es visible en los bloques con acceso estándar.
Valor predeterminado	Valor que permite predeterminar determinadas variables en la interfaz del bloque lógico o bien valor de una constante local. La indicación del valor predeterminado es opcional para variables. Si no se especifica ningún valor, se utilizará el valor predefinido para el tipo de datos indicado. Por ejemplo, el valor predefinido para BOOL es *false*. El valor predeterminado de una variable se aplica como valor de arranque en el respectivo bloque de datos de instancia. Los valores aplicados pueden sustituirse en el bloque de datos de instancia por los valores de arranque específicos de la instancia. Las constantes siempre tienen el valor predeterminado que se declaró en la interfaz del bloque. No se visualizan en los bloques de datos de instancia y por tanto tampoco se les puede asignar valores específicos de la instancia.

Remanencia	Marca una variable como remanente. Los valores de variables remanentes se conservan tras desconectar la alimentación. Esta columna solo es visible en la interfaz de bloques de función con acceso optimizado.
Visible en HMI	Indica si una variable está visible en la lista de selección de HMI mediante un ajuste predeterminado.
Accesible desde HMI	Indica si HMI puede acceder a esta variable en tiempo de ejecución.
Valor de ajuste	Marca una variable como valor de ajuste. Los valores de ajuste son valores que requieren un ajuste fino en la puesta en marcha. Esta columna solo existe en la interfaz de bloques de función.
Comentario	Comentario para documentar el elemento.

Se van a realizar algunos ejemplos sencillos para analizar el tratamiento de las funciones en SCL. Se deberán utilizar el simulador y las tablas de observación del PLC.

En los dos primeros ejercicios se van a usar funciones (FC), pero también se podrían utilizar bloques de función (FB). La diferencia es que el FB lleva asociado un DB.

Ejercicio 1

En este ejercicio se trata de pasar a una función el valor de dos marcas de palabra. La función deberá multiplicar las dos marcas y guardarlas en una variable local a la función.

Se van a utilizar dos marcas que se deberán pasar a la función. Por ello se definen dos variables del PLC en la tabla de variables: Marca_1: MW0 y Marca_2: MW2.

En la función se declaran dos datos de entrada como enteros: Dato_1 y Dato_2. También una variable local (Temp): Resultado. En la Figura 188 se incluye la interfaz de la función con la declaración de las variables (parámetros formales).

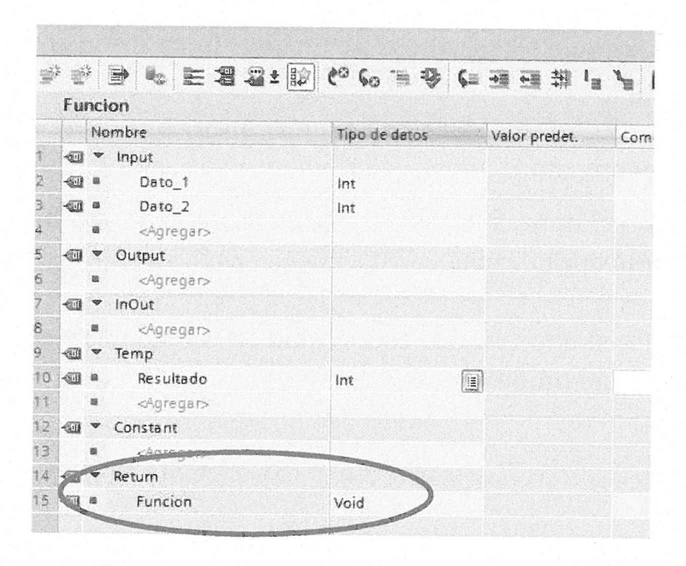


Figura 188

Hay que observar que la función no retorna ningún valor y por ello hay que poner el tipo de dato VOID. Esto es así siempre por defecto, así que, si la función no devuelve ningún valor, no hay que hacer nada en la interfaz de la función respecto al retorno de datos.

El programa de la función FC1 es:

*#Resultado := #Dato_1 * #Dato_2;* //El resultado se puede visualizar poniendo

//las gafitas.

Desde el OB1 se realiza la llamada a la función. Se deberán colocar los parámetros actuales (Marca_1 y Marca_2, declaradas en la tabla de variables del PLC y relacionadas con MW0 y MW2, respectivamente). Para realizar la llamada se debe utilizar *Drag & Drop* (arrastrar la función desde el árbol de archivos de la izquierda y soltar en el programa):

"Funcion"(Dato_1:="Marca_1",

Dato_2:="Marca_2");

Para poder comprobar el programa hay que dar valores a las marcas y eso se puede hacer desde la tabla de observación forzando los valores que se deseen, tal como se aprecia en la Figura 189.

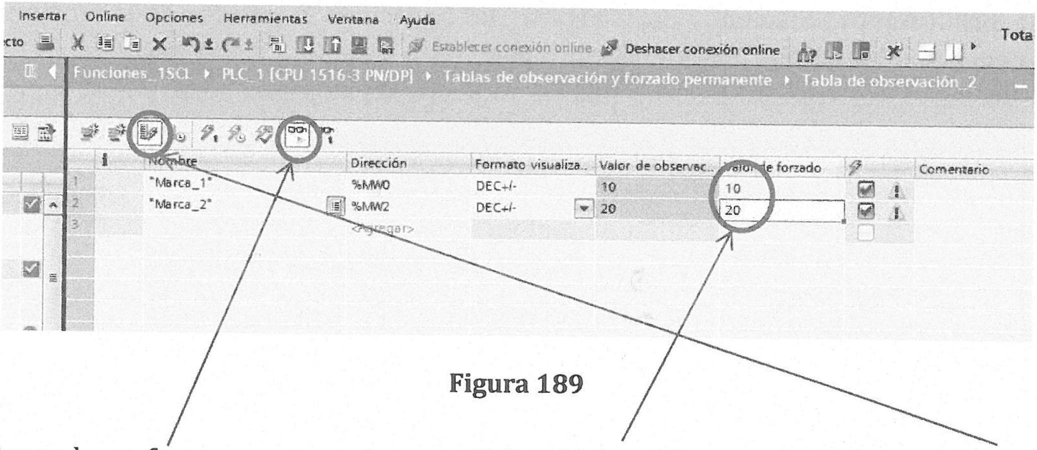


Figura 189

Se accionan las **gafas** y se ponen valores en **Valor de forzado**. Luego se **activa el forzado**.

Desde la función online (colocando las gafas) se puede observar el valor del resultado, tal como se muestra en la Figura 190.

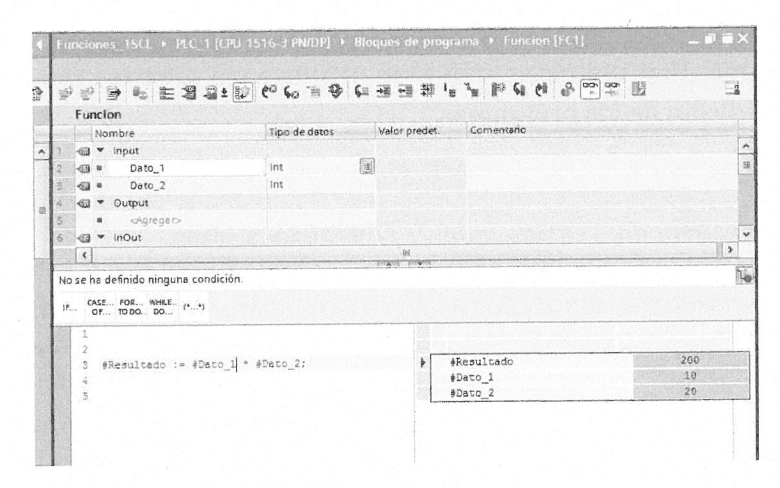


Figura 190

Ejercicio 2

Se va a realizar un ejercicio que utilice una función (FC) a la que se le pasen dos datos y los sume, el resultado de la suma los multiplica por dos y los deja en una marca. La suma la realiza en la función y la multiplicación la realiza en el bloque invocante (OB1).

Lo primero que se hará será crear la función, asignarle un nombre y crear los parámetros. Los parámetros de los datos a sumar se declararán en la interfaz de la función como entrada. El resultado se declara como local, por ejemplo, se puede incluir como Temp. Como además tiene que devolver el resultado de la suma al OB1, habrá que declarar el tipo del valor de retorno en la interfaz. Para ello, en Return se debe poner el tipo de dato que va a devolver, en nuestro caso un entero. La interfaz de la función quedará según aparece en la Figura 191.

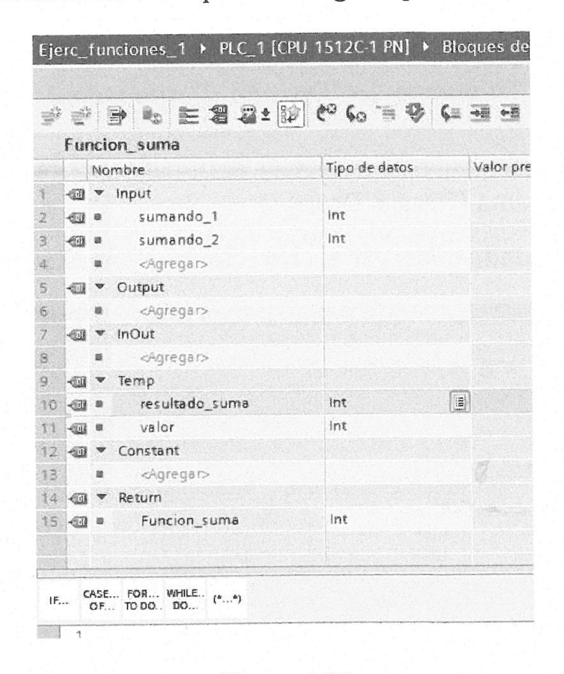


Figura 191

Como la función retorna un valor, hay que indicar en Return el tipo de dato que va a devolver, en este caso entero.

Se pueden arrastrar (Drag & Drop) las variables desde la interfaz del bloque hasta la zona de edición del programa. Hay que acostumbrarse a utilizar el arrastre porque muchas veces facilita la labor de edición de los programas.

El programa de la función es:

#resultado_suma:= #sumando_1+#sumando_2; // Realiza la suma.

#Funcion_suma := #resultado_suma; //Lo asigna a la función. Ahora Funcion_suma
// será como una variable cuyo valor es el
//resultado de la suma.

Se deben declarar las variables de entrada del PLC que se van a utilizar como sumandos y que se pasarán a la función. También se necesita una marca de palabra para guardar el resultado del producto. Se declaran en **Variables PLC**, en la tabla de variables estándar.

En la Figura 192 se adjunta dicha tabla:

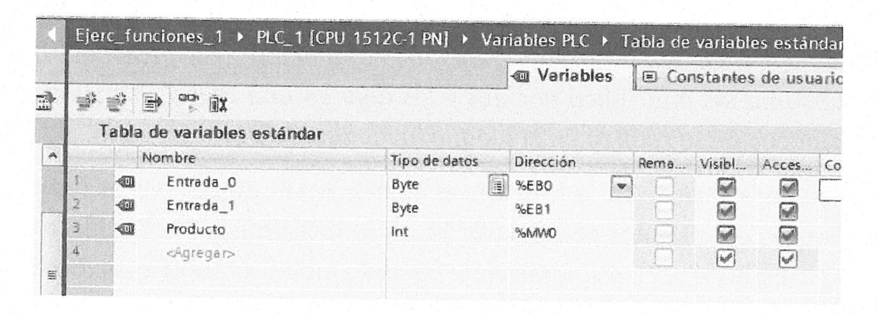


Figura 192

El programa principal (OB1) quedará así:

*"Producto":="Funcion_suma"(sumando_1:="Entrada_0",sumando_2:="Entrada_1")*2;*

Para colocar la función, es conveniente arrastrar la función desde el árbol del proyecto hasta la zona de edición del programa (Drag & Drop).

Si se abre el simulador, se puede observar el correcto funcionamiento del programa:

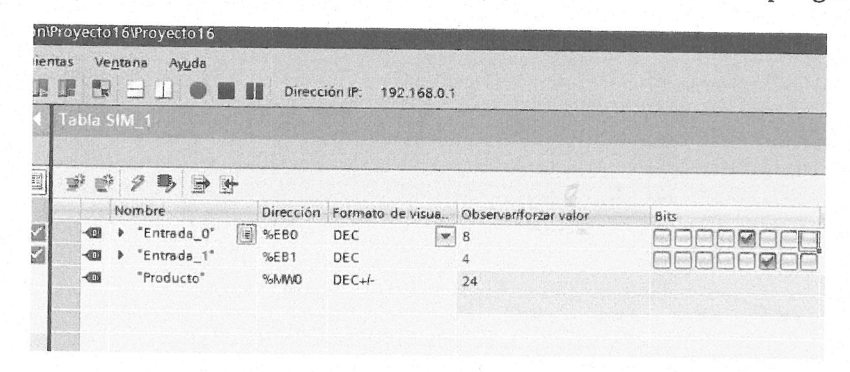


Figura 193

Ejercicio 3

En el siguiente ejemplo se va a trabajar con matrices y funciones. La función va a recibir como parámetros actuales una matriz unidimensional de 8 elementos. Esta matriz se va a inicializar con el valor de su elemento más uno. Por ejemplo, el elemento 0 de la matriz hay que inicializarlo con el valor 1, el elemento 4 con el valor 5, el elemento 5 con el vàlor 6 y así sucesivamente. La función multiplicará por 10 cada elemento de la matriz original, dejando el resultado en otra matriz.

Desde el OB1 se pasarán los parámetros actuales a la función. Lo primero que se deberá hacer en el OB1 es crear una matriz unidimensional de 8 elementos y luego inicializar cada elemento de la misma como indica el enunciado. La matriz se va a crear en la interfaz del OB1 como una matriz temporal. Igualmente se va a utilizar una variable de tipo entero como índice de la matriz (Figura 194). Se llama *m* a la matriz e *índice* a la variable temporal que va a hacer de índice.

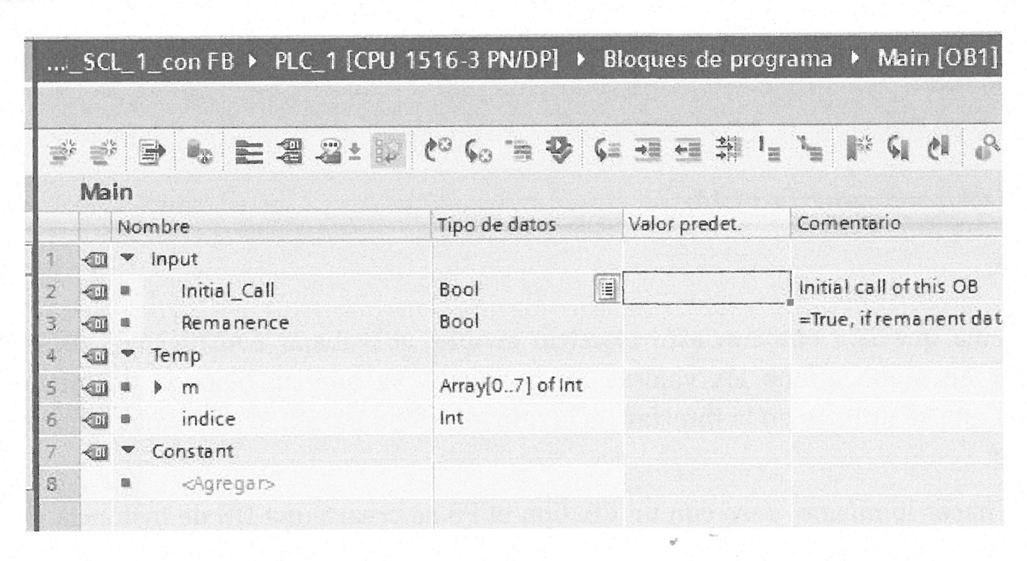


Figura 194

El programa del OB1 es el siguiente:

```
FOR #indice := 0 TO 7 DO    // Se recorre cada elemento de la matriz para
  #m[#indice] := #indice + 1;      // sumar 1 a cada elemento.
END_FOR;
"Funcion"(#m);      // Se realiza la llamada a la función pasándole la matriz m.
```

La función tiene los parámetros formales «matriz» e «i», como matriz e índice respectivamente. El parámetro «matriz» como entrada y el parámetro «i" como temporal. En la Figura 195 se incluyen ejemplos.

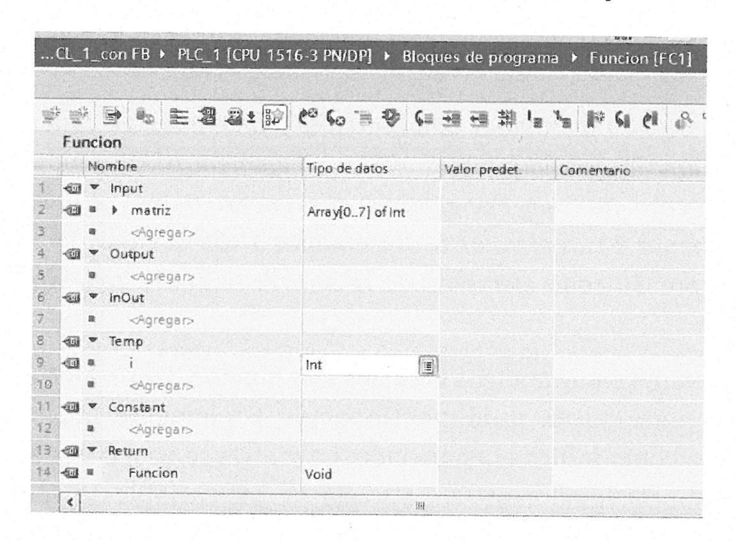


Figura 195

Y el programa de la función que recibe la matriz del OB1 y la multiplica (elemento a elemento) es el siguiente:

FOR #i := 0 TO 7 DO

*#matriz[#i] := #matriz[#i] * 10;*

END_FOR;

El problema que se tiene con este ejercicio es que, al trabajar con una FC, no se pueden visualizar en modo online los valores de la matriz, ya que no dispone de memoria. Las variables que se utilizan en la interfaz de la función como parámetros formales no se pueden visualizar.

Vamos a hacer lo mismo, pero con un FB. Con el FB se creará una DB de instancia y desde el DB se puede visualizar toda la matriz en modo online.

Ahora lo que se ha hecho en la función se hace en el FB. Lo único que cambia es que la declaración de la matriz se va a guardar en la memoria (en el DB) y ya se podrá visualizar su valor online. En el OB1 se hará la llamada al FB arrastrándola desde el árbol del proyecto. Cuando se coloque, pedirá crear el DB, que se aceptará sin más.

Para poder visualizar la matriz original, la que se ha inicializado en el OB1, se va a crear en la función otra matriz que se le va a llamar matriz_original. En ella se guardará la matriz inicializada en el OB1. Esta FB tendrá tres parámetros: dos matrices como INPUT (matriz y matriz_original) y el índice (i), como TEMP.

Primero creamos el FB (desde la opción de agregar un nuevo bloque). Se escriben los parámetros y las instrucciones. Después se abre el OB1 y se crean sus parámetros (m, i) igual que se ha hecho con la función. Se llama al FB, aceptando el DB que se solicita y se pasan los parámetros actuales al FB. En este caso se pasa la matriz creada (m) a las dos matrices del FB.

Las siguientes figuras (fig. 196 y Fig. 197) incluyen ejemplos de parámetros del OB1 y del FB, respectivamente.

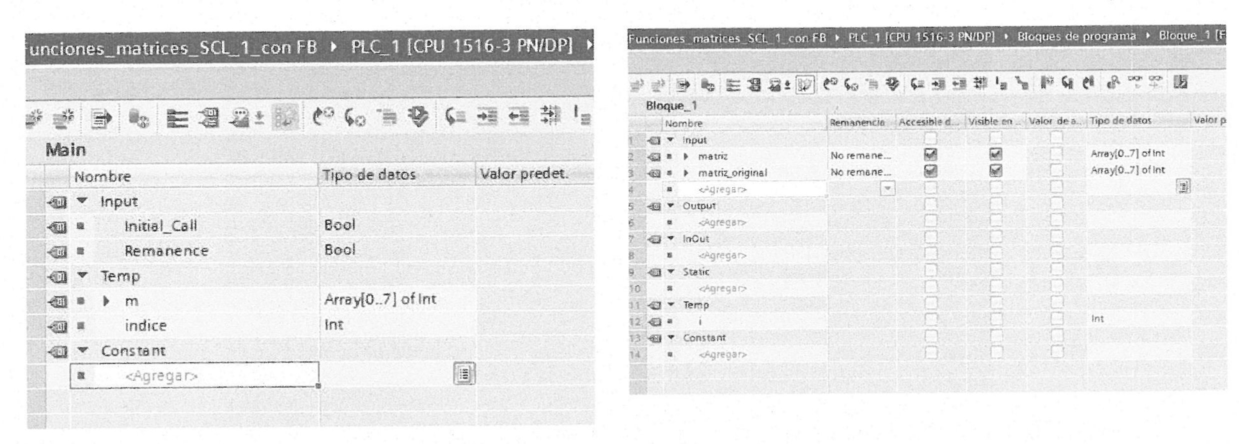


Figura 196 **Figura 197**

Los programas son los siguientes:

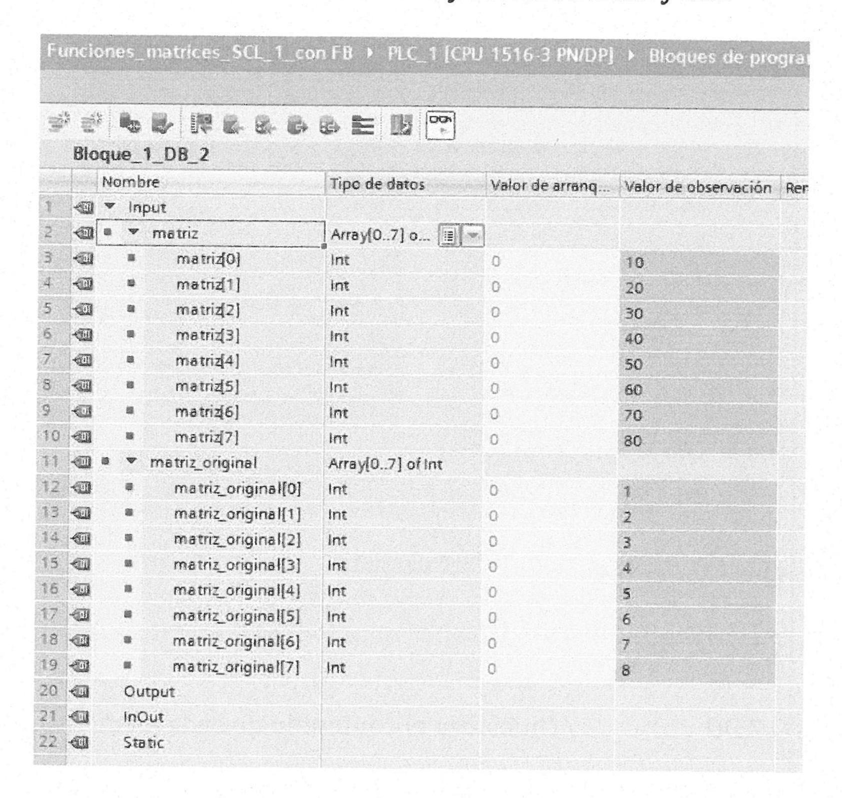
Programa del OB1

```
FOR #indice := 0 TO 7 DO                //Inicializa el contenido de cada elemento de la matriz
  #m[#indice] := #indice + 1;           //sumándole uno a cada elemento y partiendo de cero.
  END_FOR;
"Bloque_1_DB_2"(matriz:=#m,             //Se llama al FB pasando la matriz m a las dos
          matriz_original:=#m);         //matrices del FB.
```

Programa del FB1

```
FOR #i := 0 TO 7 DO   //Multiplica por 10 cada elemento de la matriz pasada desde el OB1.
  #matriz[#i] := #matriz[#i] * 10;
END_FOR;
```

En la Figura 198 se incluye el DB con el resultado visualizado al poner las gafas.



Funciones_matrices_SCL_1_con FB ▸ PLC_1 [CPU 1516-3 PN/DP] ▸ Bloques de progra

Bloque_1_DB_2

	Nombre	Tipo de datos	Valor de arranq...	Valor de observación	Rer
1	▼ Input				
2	▼ matriz	Array[0..7] o...			
3	matriz[0]	Int	0	10	
4	matriz[1]	Int	0	20	
5	matriz[2]	Int	0	30	
6	matriz[3]	Int	0	40	
7	matriz[4]	Int	0	50	
8	matriz[5]	Int	0	60	
9	matriz[6]	Int	0	70	
10	matriz[7]	Int	0	80	
11	▼ matriz_original	Array[0..7] of Int			
12	matriz_original[0]	Int	0	1	
13	matriz_original[1]	Int	0	2	
14	matriz_original[2]	Int	0	3	
15	matriz_original[3]	Int	0	4	
16	matriz_original[4]	Int	0	5	
17	matriz_original[5]	Int	0	6	
18	matriz_original[6]	Int	0	7	
19	matriz_original[7]	Int	0	8	
20	Output				
21	InOut				
22	Static				

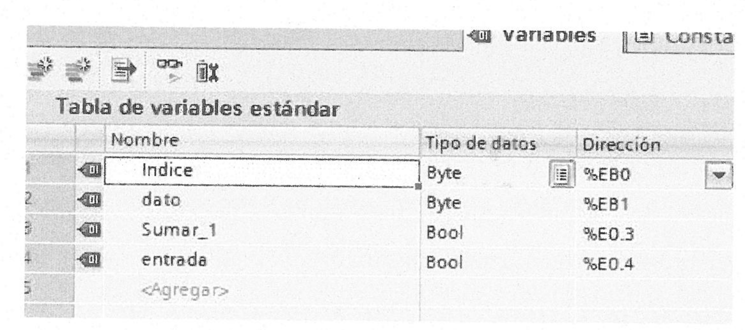
Figura 198

Ejercicio 4

En este ejercicio se siguen utilizando funciones y matrices, pero esta vez se introducen variables del PLC. Se va a volver al ejemplo realizado en el apartado de matrices. Se recuerda su planteamiento: se dispone de tres entradas del PLC, E0.0, E0.1 y E0.2, con las que se selecciona un elemento de una matriz unidimensional de 8 elementos. Con el byte de entradas EB1 se introduce un dato a dicha matriz. Con la entrada E0.3 se suma 1 a cada elemento de la matriz. La entrada E0.4 es la que acepta que el valor de EB1 se pase a la matriz, a la posición indicada por el valor de las entradas E0.0, E0.1 y E0.2.

En este caso se va a utilizar un segundo FB al que se le pasará la matriz original y en este nuevo FB se le sumará uno al contenido de cada elemento de la matriz. Se utilizará multiinstancia para que en un mismo DB se tengan todos los datos, tanto del FB1 como del FB2.

Se crean las mismas variables del PLC que en el ejercicio dos de matrices, como se aprecia en la Figura 199.



Variables | Consta

Tabla de variables estándar

	Nombre	Tipo de datos	Dirección
1	Indice	Byte	%EB0
2	dato	Byte	%EB1
3	Sumar_1	Bool	%E0.3
4	entrada	Bool	%E0.4
5	<Agregar>		

Figura 199

A continuación se crean los parámetros del FB1 (Figura 200).

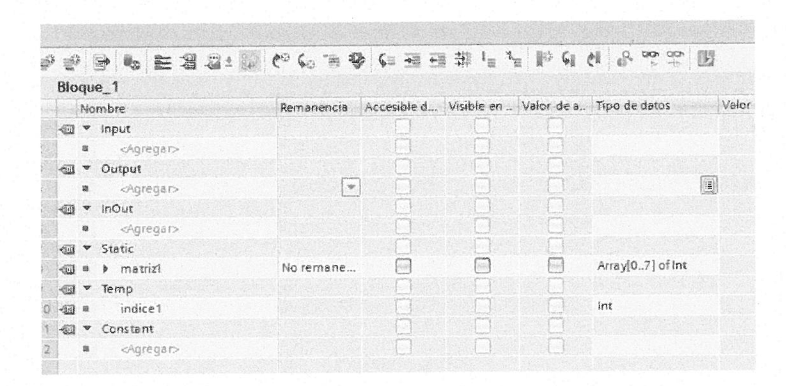


Figura 200

Se realiza el programa del FB1, que es el que tiene que cargar los datos en la matriz original con la entrada EB1, seleccionado por el índice que lo forma las entradas E0.0, E0.1 y E0.2. La entrada E0.4 es la que determina el momento en que se hace efectiva la carga en la matriz. Este programa es el mismo en la parte de carga y selección donde se carga.

Aquí también se debe hacer la llamada al FB2, que es la encargada de recibir la matriz original y sumarle uno a cada elemento. Pero antes hay que crear el bloque de función 2, FB2, con sus parámetros. Dichos parámetros se pueden observar en la Figura 201.

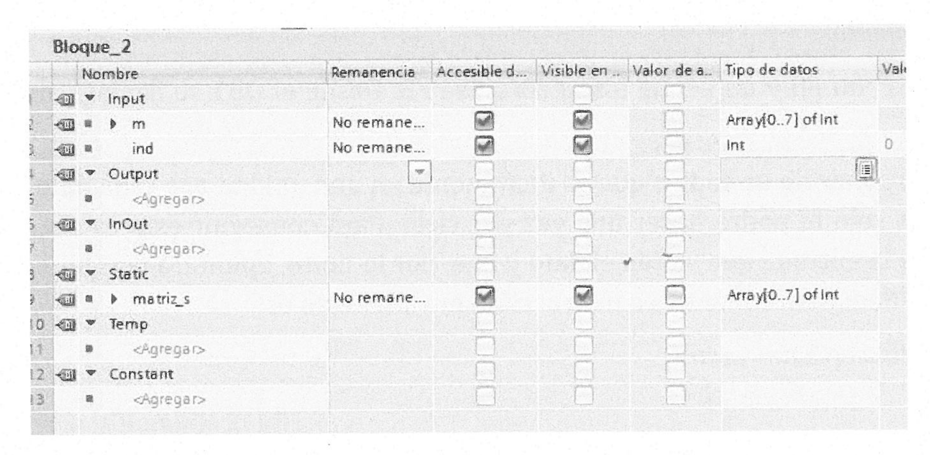


Figura 201

Después se realizan las llamadas a FB1 en el OB1 y a FB2 desde el FB1. Se debe recordar que es útil emplear el método de arrastre desde el árbol del proyecto. Cuando se realice la llamada al FB2 desde el FB1, habrá que seleccionar la opción multiinstancia. De este modo ambos bloques, FB1 y FB2, utilizarán la misma DB.

Los programas quedan del siguiente modo:

OB1:

"Bloque_1_DB"();

FB1:

#indice1 := "Indice" AND 7;

```
IF "entrada" = 1 THEN
  #matriz1[#indice1] := "dato";
END_IF;
IF "Sumar_1"=1 THEN
#Bloque_2_Instance(m:=#matriz1,
          ind:=0);
END_IF;
```

FB2:

```
  FOR #ind := 0 TO 7 DO
    #matriz_s[#ind] := #m[#ind] + 1;
  END_FOR;
```

Si se abre el DB, y se ponen las gafas para observar las variables de las matrices, se puede observar el resultado.

Se debe realizar un programa en el que una matriz unidimensional de 8 elementos se vaya cargando con el dato que tenga la entrada de byte EB1. El dato se valida con la entrada de bit E0.0. Cada vez que se cargue un valor en la matriz, se deberá apuntar automáticamente al siguiente elemento.

Se va a utilizar un FB y un DB de instancia a ese FB. Desde el OB1 se llamará al FB de forma permanente.

Se deberá disponer de un índice que se incremente en una unidad por cada dato introducido. Además, esto solo lo podrá hacer una vez por ciclo. Para conseguir esto es necesario que la activación de la entrada que valida el dato y que, por lo tanto, apuntará al siguiente elemento, se active por flanco ascendente. La instrucción que hace esto es:

```
#R_TRIG_Instance(CLK:="input",
                 Q=>#salida);
```

Donde input es la entrada donde se debe detectar el flaco ascendente y Q la salida que se active por cada ciclo. Esta instrucción utiliza un DB de instancia, ya que en realidad es un bloque de función, no una instrucción.

La tabla de variables del PLC y de la interfaz del FB1 se incluyen en las Figuras 202 y 203, respectivamente.

Variables | Constantes de usuario | Constantes d

Tabla de variables estándar

	Nombre	Tipo de datos	Dirección	Rema...	Visibl...	Acces...
1	entrada_valor	Byte	%EB1		✓	✓
2	validacion_entrada	Bool	%E0.0		✓	✓
3	<Agregar>				✓	✓

Figura 202

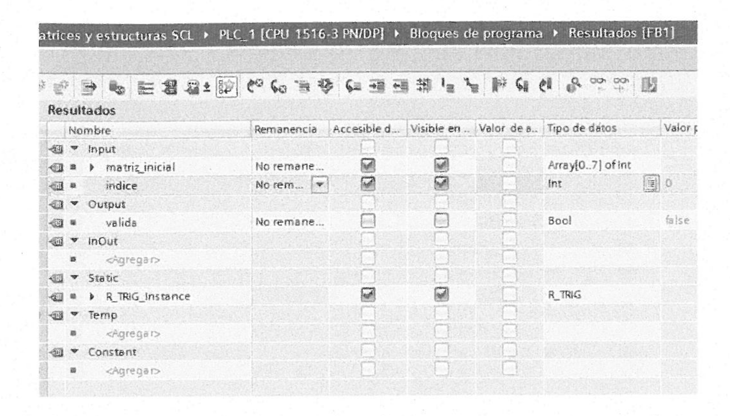


Figura 203

El programa del FB es el siguiente:

```
#R_TRIG_Instance(CLK:="validacion_entrada",     // Detecta flanco ascendente y valida"
                 Q=>#valida);                    // solo se pone a uno una vez por ciclo.
IF #indice<=7 AND #valida THEN                  //Si el índice es menor de 7 (la matriz es de 8
   #matriz_inicial[#indice] := "entrada_valor";  //elementos 0...7) y valida =1, entonces...
   #indice := #indice + 1;                       //se carga el valor en el elemento actual y
                                                 // el índice apunta al siguiente elemento
   IF #indice>7 THEN                             // Si ya se han rastreado los 8 elementos, se inicializa
#indice := 0;                                    // el índice.
   END_IF;
   END_IF;
```

El OB1 solo hace la llamada al FB1:

```
"Resultados_DB"();   //"Resultados" es el nombre del FB1.
```

Ejercicio 5

Al ejercicio anterior se le van a añadir dos funciones (), una que realiza la raíz cuadrada y otra el cuadrado de cada uno de los elementos de la matriz original. Los resultados se guardarán en una matriz estructurada en resultados_raiz y en resultados_cuadrado. Cada función realizará una operación: la raíz, la función_raiz, y el cuadrado, la función_cuadrado.

Como se indica en el enunciado, se debe crear una matriz de estructuras con dos variables: raíz y cuadrado, ambas de tipo entero.

Para crear este tipo de matrices se debe ir a la interfaz del FB y crear una matriz de 8 elementos de tipo STRUCT. Dentro del primer elemento se crean, en este caso, dos variables

de tipo entero, raíz y cuadrado. Ahora aparecerán esas dos variables en todos los elementos de esa matriz (Figura 204).

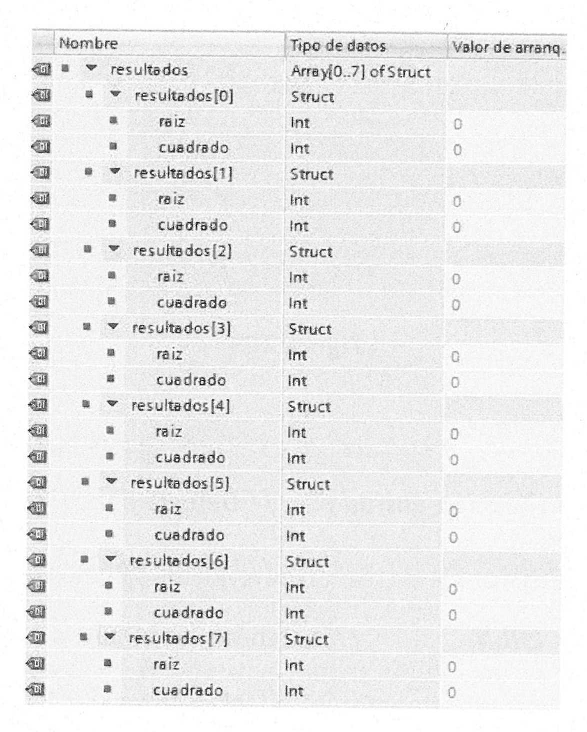


Figura 204

Para acceder a cada elemento se debe indicar el nombre de la matriz y, a continuación, separado por un punto (.) el nombre de la variable de la estructura. Por ejemplo: Resultado[2].raiz, que hace referencia al elemento 2 de la variable raíz perteneciente a la matriz resultado.

En las Figuras 205, 206, 207 y 208 se pueden apreciar las variables del PLC y los parámetros del FB y las funciones, respectivamente.

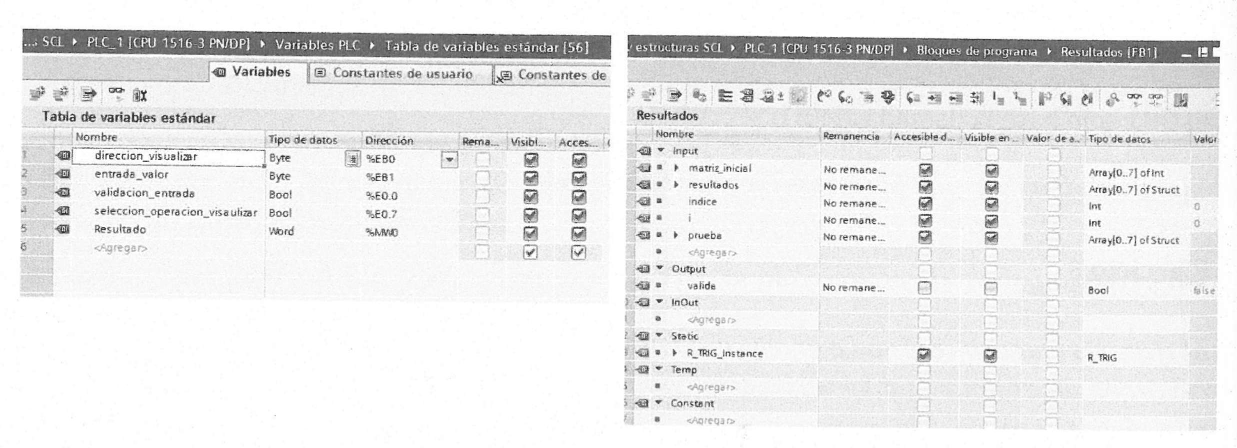


Figura 205 **Figura 206**

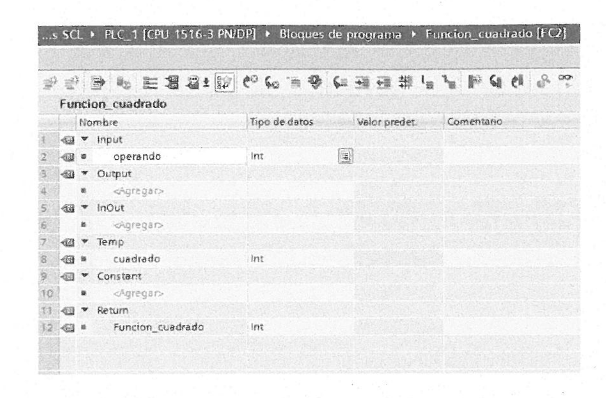


Figura 207

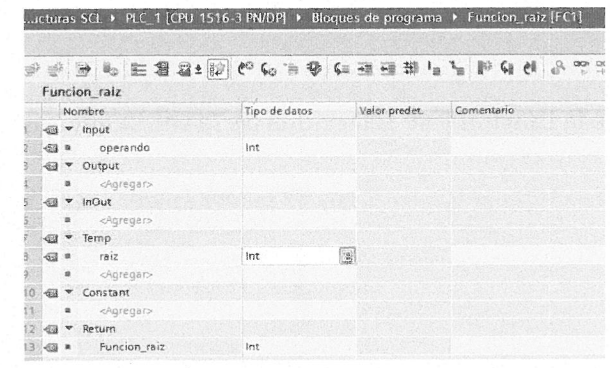


Figura 208

El programa del FB es el siguiente:

```
// La primera parte es lo mismo que el ejercicio anterior.
#R_TRIG_Instance(CLK:="validacion_entrada",
                 Q=>#valida);
IF #indice<=7 AND #valida THEN
  #matriz_inicial[#indice] := "entrada_valor";
    #indice := #indice + 1;
  IF #indice>7 THEN
    #indice := 0;
  END_IF;
END_IF;
```

//En esta segunda parte es donde se llaman a las funciones cuadrado y raíz, y se guardan los resultados en la matriz de estructura. Cada vez que se modifique algún valor en la matriz inicial, se realizarán los cálculos. Los cálculos se realizan continuamente. Las dos funciones devuelven el valor del resultado.

```
FOR #i := 0 TO 7 DO
    #resultados[#i].cuadrado := "Funcion_cuadrado"(#matriz_inicial[#i]);
    #resultados[#i].raiz:="Funcion_raiz"(#matriz_inicial[#i]);
END_FOR;
```

Los programas de las funciones son los siguientes.

Función cuadrado (SQR):	Función raíz (SQRT):
#cuadrado := SQR(#operando); *#Funcion_cuadrado := #cuadrado;*	*#raiz:= SQRT(#operando);* *#Funcion_raiz := #raiz;*

Para colocar la función y asignarle el valor que va a devolver, hay que coger la función con valor int, tal como indica la Figura 209. En este caso no se debe arrastrar, se pone el nombre de la función y se selecciona de **la tabla** que sale.

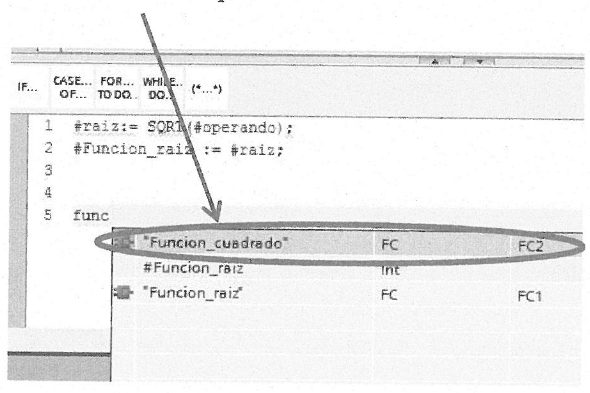


Figura 209

El OB1 es el mismo que el ejercicio anterior; solo lleva la llamada al FB: *"Resultados_DB"();*

Para comprobar, se debe abrir el DB de instancia del FB y poner las gafas para ver los contenidos de las matrices.

En este ejercicio se pretende cargar una matriz con los datos de la entrada de byte EB1 y ordenarla de mayor a menor. Con la entrada de bit E0.0 se valida la entrada de datos como en el ejercicio anterior. Con la activación de la entrada E0.1, se realizará la ordenación de la matriz y se deja el resultado en otra matriz.

Para ordenar una matriz, hay varios métodos. Se va a utilizar el método *Bubble Sort.* En este método se comparan todos los elementos con todos. Cuando se cumple la condición de comparación, se intercambian sus valores. Si no se cumple la condición, se dejan los valores tal cual están y se pasa al siguiente elemento. En el siguiente ejemplo se tiene un *array* de 4 elementos. Se desea ordenar de mayor a menor.

Valor	23	3	5	87
Elemento	[0]	[1]	[2]	[3]

Se empieza a comparar por elemento 0. Se compara con el siguiente ¿23<3?, la respuesta es no, entonces se deja cada uno en su lugar y se compara con el siguiente elemento.

Valor	23	3	5	87
Elemento	[0]	[1]	[2]	[3]

Ahora pasa lo mismo, ya que 5 es menor que 23, no se intercambian los valores. Se pasa al siguiente elemento:

Valor	23	3	5	87
Elemento	[0]	[1]	[2]	[3]

El valor 23 es menor que 87 y entonces se intercambian los valores entre los elementos:

Valor	87	3	5	23
Elemento	[0]	[1]	[2]	[3]

El primer elemento ya ha sido comparado con todos los demás y se debe pasar al siguiente. Ahora el **elemento 1 se compara con todos los siguientes.**

Valor	87	3	5	23
Elemento	[0]	[1]	[2]	[3]

El valor 5 es mayor que 3, así que se deben intercambiar los dos valores:

Valor	87	5	3	23
Elemento	[0]	[1]	[2]	[3]

Se sigue con el siguiente elemento:

Valor	87	5	3	23
Elemento	[0]	[1]	[2]	[3]

En este caso, también 23 es mayor que 5 y se intercambiarán los valores:

Valor	87	23	3	5
Elemento	[0]	[1]	[2]	[3]

El elemento [1] ya ha sido intercambiado por todos los demás y se pasa al siguiente:

Valor	87	23	3	5
Elemento	[0]	[1]	[2]	[3]

El valor 5 es mayor que el 3 y se intercambian los datos:

Valor	87	23	5	3
Elemento	[0]	[1]	[2]	[3]

Ahora todos los elementos han sido comparados con todos y la matriz ya está ordenada.

En base a este método se desea ordenar la matriz del ejercicio.

Se va a utilizar un FB llamado ENTRADA donde se introducen los datos de la entrada EB1 con cada activación de E0.0. La matriz va apuntando al siguiente elemento para llenarla de datos. Cada vez que hay un dato nuevo en la matriz de entrada, se llama a otro FB para ordenar la matriz de mayor a menor. La ordenación se deja en otra matriz denominada MATRIZ_ORDENADA.

En las siguientes figuras se incluyen las variables del PLC, del FB de entrada y la FB de ordenación.

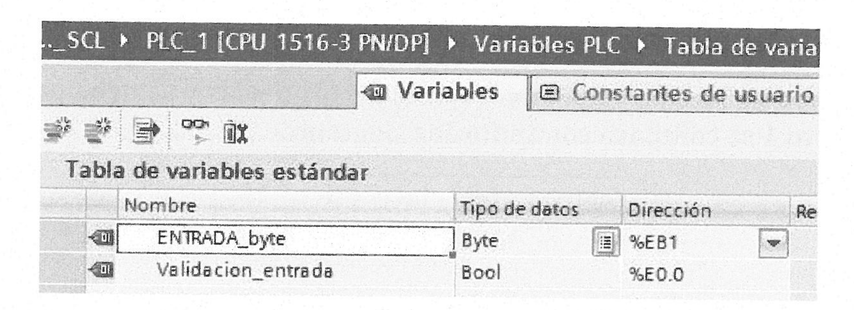


Figura 210

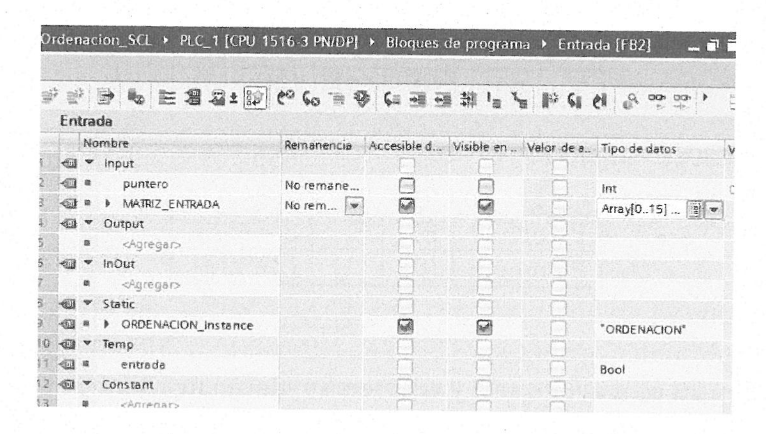


Figura 211

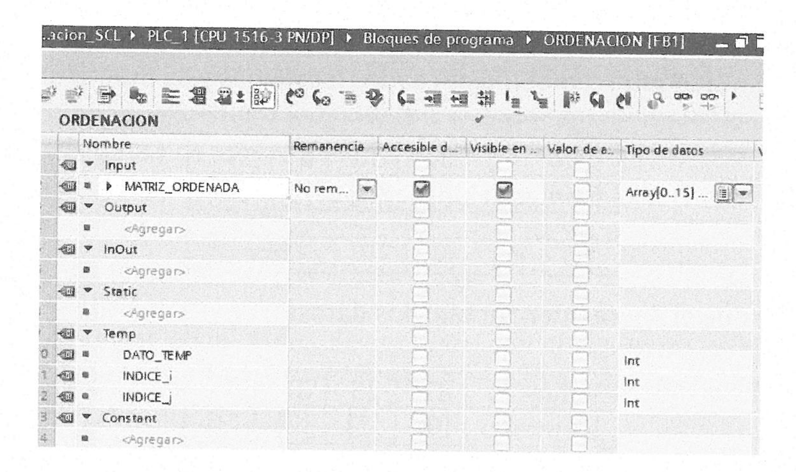


Figura 212

Primero se crea el FB de entrada con el siguiente programa:

```
"R_TRIG_DB_1"(CLK := "Validacion_entrada",        //Detección de flanco ascendente de E0.0.
              Q => #entrada);
IF #entrada AND #puntero <= 15 THEN               //Entrada de datos en la matriz.
  #MATRIZ_ENTRADA[#puntero] := "ENTRADA_byte";
```

```
    #puntero := #puntero + 1;
    IF #puntero > 15 THEN
        #puntero := 0;
    END_IF;
END_IF;
#ORDENACION_Instance(#MATRIZ_ENTRADA);  // Llamada al FB de ordenación.
```

Esta llamada se debe poner una vez creado el FB de ordenación. Cuando pida el DB, se puede seleccionar de instancia o multiinstancia. Para poder visualizar los resultados es más cómodo elegir multiinstancia.

El programa del FB de ordenación es el encargado de realizar dicha ordenación de mayor a menor. El siguiente programa está basado en el procedimiento de ordenación *Bubble Sort* que ya hemos visto.

El índice j es el que determina el elemento a comparar con todos los demás. El índice i es el que se va moviendo para rastrear cada elemento a partir del índice j.

La comparación siempre es entre el elemento del índice j con el i+1, es decir, el elemento actual con cada uno de los siguientes, hasta el último elemento.

Cuando se cumpla la condición, se deben intercambiar los valores de los dos elementos. Para ello, es necesario disponer de una variable intermedia que guarde temporalmente uno de los valores. Es el caso de la variable DATO_TEMP.

```
FOR #INDICE_j := 0 TO 14 DO
    FOR #INDICE_i := #INDICE_j TO 14 DO
        IF #MATRIZ_ORDENADA[#INDICE_j] < #MATRIZ_ORDENADA[#INDICE_i + 1] THEN
            #DATO_TEMP := #MATRIZ_ORDENADA[#INDICE_j];
            #MATRIZ_ORDENADA[#INDICE_j] := #MATRIZ_ORDENADA[#INDICE_i + 1];
            #MATRIZ_ORDENADA[#INDICE_i + 1] := #DATO_TEMP;
                END_IF;
    END_FOR;
END_FOR;
```

El programa Ob1 realiza la llamada al FB de entrada:

```
"Entrada_DB"();
```

Para comprobar los resultados, se deben poner las gafas en el DB de entrada, donde se verán las instancias del FB de entrada y del FB de ordenación y, por lo tanto, la matriz de entrada y la matriz ordenada.

EJEMPLOS VARIOS EN SCL

GARAJE

Se trata de realizar el programa de un garaje para 10 coches que tiene un acceso de entrada y otro de salida con sendas barreras. Si se llena, no se permitirá la entrada de más coches hasta que no salga alguno. Existen dos sensores, tanto en la entrada como en la salida. En la entrada, uno de los sensores, situado antes de la barrera, detecta que hay un coche que desea entrar. Otro sensor está situado después de la barrera y detecta que el coche ya ha entrado. La barrera de entrada se sube cuando se detecta coche. La barrera baja de forma automática dos segundos después de haber entrado el coche. Los sensores están situados a una distancia de menos de un metro uno de otro.

En la salida sucede lo mismo, existen dos sensores: uno para detectar que un coche quiere salir y otro, situado a menos de un metro, para detectar que el coche ya ha salido.

Lo primero que se debe hacer es organizar las entradas y salidas necesarias. Es conveniente asignar nombres que tengan relación con la función que desempeñan:

ENTRADAS		SALIDAS	
BEabajo	E0.0	BarreraEsubir	A0.0
BEarriba	E0.1	BarreraEbajar	A0.1
DE1	E0.2	BarreraSsubir	A0.2
DE2	E0.3	BarreraSbajar	A0.3
BSabajo	E0.4		
BSarriba	E0.5		
DS1	E0.6		
DS2	E0.7		

Otros:
Temporizador TON Barrera de entrada
Temporizador TON Barrera de salida
Marca coche dentro: M0.0
Marca coche fuera: M0.1
Marca de lleno: M0.2

Aclaración de los dos sensores, tanto para la entrada como para la salida:

El detector 1 (DE1 o DS1) hace que la barrera se levante cuando detecta coche. Los detectores que se encuentran después de la barrera (DE2 y DS2) sirven para saber, junto a los detectores DE1 o DS1, que el coche ya ha pasado y asegurarse de que la barrera nunca se cerrará si el coche aún no ha pasado. Cuando un coche entra o cuando sale, durante un momento los dos sensores estarán activados porque ambos detectarán coche. Solo cuando ninguno de los dos detecte coche, se bajará la barrera.

El programa en SCL es el siguiente:

```
(*Si se detecta coche en la entrada y el garaje no está lleno, sube la barrera de entrada*)
IF "De1"=1 AND "Lleno"=0 THEN
  "BarreraEsubir" := 1;
END_IF;
(*Cuando la barrera está arriba, se para*)
IF "BEarriba" = 1 THEN
  "BarreraEsubir" := 0;
END_IF;
(*Cuando la barrera está arriba y no detecta coche ninguno de los dos detectores, arranca el
temporizador de 2 s.
Cuando la barrera empiece a bajar el detector BEarriba, ya no estará activado y el
temporizador TON se pone a cero, por lo que Q también.*)
"IEC_Timer_0_DB".TON(IN:="BEarriba"=1 AND "De1"=0 AND "De2"=0,
                     PT:=t#2s,
                     Q=>"CocheDentro");
(*Cuando termina la temporización, baja la barrera de entrada y se setea BarreraEbajar, por el
motivo explicado en el comentario anterior*)
 IF "CocheDentro"=1 THEN
   "BarreraEbajar" := 1;
 END_IF;
 (*Cuando la barrera ha bajado, se para. También se resetea el indicador de Coche Dentro*)
IF "BEabajo" = 1 THEN
  "BarreraEbajar" := 0;
  "CocheDentro" := 0;
END_IF;
(*Si se detecta coche en la salida, sube la barrera de salida*)
IF "Ds1" = 1 THEN
  "BarreraSsubir" := 1;
END_IF;
(*Cuando la barrera está arriba, se para*)
IF "BSarriba" = 1 THEN
  "BarreraSsubir" := 0;
END_IF;
```

(*Cuando la barrera está arriba y no detecta coche ninguno de los dos detectores, arranca el temporizador de 2 s. Cuando la barrera empiece a bajar, el detector BSarriba ya no estará activado y el temporizador TON se pondrá a cero, por lo que Q también *)

```
"IEC_Timer_0_DB_1".TON(IN:="BSarriba"= 1 AND"Ds1"= 0 AND"Ds2" = 0,
                       PT:=t#2s,
                       Q=>"CocheFuera");
```

(*Cuando termina la temporización, baja la barrera de salida y se setea BarreraSbajar, por el motivo explicado en el comentario anterior.*)

```
IF "CocheFuera" = 1 THEN
  "BarreraSbajar" := 1;
END_IF;
```

(*Cuando la barrera ha bajado, se para. También se resetea el indicador de "Coche Fuera"*)

```
IF "BSabajo" = 1 THEN
  "BarreraSbajar" := 0;
  "CocheFuera" := 0;
END_IF;
```

(*Los detectores de coche dentro y coche fuera son los que incrementan y decrementan respectivamente el contador. Cuando llega a 10, se activa la variable de lleno e impide subir la barrera de entrada hasta que salga un coche*)

ADQUISICIÓN Y CLASIFICACIÓN DE VALORES

En este ejercicio se trabaja con el uso de matrices y funciones. Se trata de introducir datos a una matriz y operar con esos datos. Posteriormente se debe sacar la información que se desea. En el ejercicio hay varios procesos:

Entrada de información.

Procesamiento de la información (cálculos y ordenación).

Salida de la solicitud de información.

Mediante ocho interruptores de entrada se selecciona un dato (EB1). El dato se valida con la activación de una entrada (E0.0) y se guarda en memoria (matriz). El dato se va guardando en un array con cada activación de la entrada de validación (E0.0), en la posición que le corresponde siguiendo un orden secuencial (de 0 a 15).

Posteriormente, y con la activación/validación de otra entrada (E0.1), se ordena el array de mayor valor a menor. Igualmente se calcula el cuadrado y la raíz cuadrada de cada elemento (con la misma activación de E0.1). Esto es el procesamiento de la información.

Como salida se tiene que seleccionar en primer lugar la posición de la matriz que se desea leer. Esto se realiza con cuatro interruptores (E0.6, E0.5, E0.4, E0.3). Otro interruptor validará la

posición que se va a leer (E0.7). El valor que se visualiza se selecciona de entre estas posibilidades:

- *Con un interruptor (E0.3) se selecciona entre visualizar el valor medido inicialmente o el cálculo realizado (cuadrado o raíz). El valor medido se lee de la matriz de entrada ya ordenada.*
- *Con otro interruptor (E0.2) (solo operativo si se ha seleccionado visualizar el cálculo realizado) se opta entre la visualización del cuadrado o la raíz.*

Otra entrada (E1.7) validará el dato de salida. Es decir, el procedimiento para extraer un valor de la memoria y dejarlo en una marca (MW0) es:

Seleccionar la dirección de memoria de donde se desea extraer el dato.

Validar dicha dirección (E0.7).

Seleccionar qué se dese sacar (E0.3/E0.2).

Validar dicho valor y sacarlo a la marca (E1.7).

Para realizar este ejercicio se van a utilizar los siguientes bloques:

OB1: entrada y salida de datos. Llamada al FB de entrada.

FB1 Entrada: recoge el dato de entrada y lo guarda en memoria. Llamada al FB de ordenación para ordenar la matriz o para recoger el dato de salida.

FB2 Ordenación (multiinstancia): ordena el array original, y crea array de operación raíz y cuadrado. También determina el valor que se va a extraer.

FC Cálculos: realiza el cálculo de la raíz cuadrada y del cuadrado de cada posición del array original.

Los símbolos utilizados en el ejercicio son:

ENTRADAS		**SALIDAS**	
entrada_byte0	EB0	Marca_salida	MW0
entrada_dato	EB1	Marca_desplaza	MW2
Valida_entrada	E0.0		
acepta_ord_calc	E0.1		
selección_raiz_cuadrado	E0.2		
selección_vmedido_vcalculado	E0.3		
Acepta_codif	E0.7		

Valida_salida	E1.7	
Otros: FB1: Entrada_datos FB2: Ordenacion_calculo_salida FC1: F_Cuadrado FC2 : F_Raiz		

Los programas de los diferentes bloques son los siguientes:

OB1

Se han definido los siguientes parámetros:

Temp:

```
dato_eb0          Byte
entrada_acepta    Bool
dato_desplaz      Byte
```

Programa

```
(*Llama de forma cíclica a la introducción de datos.*)
"Entrada_datos_DB"();
(*Selecciona la dirección de donde se sacará el dato deseado.*)
"R_TRIG_DB_2"(CLK := "acepta_direcc_salida",
        Q => #entrada_acepta);
IF #entrada_acepta THEN
  #dato_eb0 := "entrada_byte0";
  #dato_eb0 := #dato_eb0 AND 2#01111000;
  #dato_desplaz := SHR(IN := #dato_eb0, N := 3);
  "marca_desplaz" := #dato_desplaz;
END_IF;
```

FB1 (Entrada_datos)

Los parámetros definidos son:

Input:

```
puntero           Int
matriz_entrada    Array[0..15] of Int
entrada_1         Bool
entrada_2         Bool
```

Programa

```
(*Carga los datos en la matriz validando con E0.0 cada entrada.*)
"R_TRIG_DB"(CLK:="valida_entrada",
        Q=>#entrada_1);
IF #entrada_1 AND #puntero<=15 THEN
  #matriz_entrada[#puntero] := "entrada_dato";
  #puntero := #puntero + 1;
  IF #puntero>15 THEN
    #puntero := 0;
  END_IF;
END_IF;
(*Con E0.1 se ordena la matriz de entrada en otra matriz (matriz_ordenada) llamando al FB de ordenación (de multiinstancia). También llama al FB de ordenación con E1.7 para extraer el valor de salida.*)
"R_TRIG_DB_1"(CLK:="acepta_ordena_calc",
        Q=>#entrada_2);
IF #entrada_2 OR "valida_salida" THEN
  #Ordenación_calculo_salida_Instance(matriz_ordenada:=#matriz_entrada);
END_IF;
```

FC1 (F_Cuadrado)

Se han declarado los siguientes parámetros:

```
Input:
operando      Int
Temp:
cuadrado      Int
Return:
F_Cuadrado    Int
```

Programa

```
#cuadrado := SQR(#operando);
#F_Cuadrado := #cuadrado;
```

FC2 (F_Raiz)

Se han declarado los siguientes parámetros:

```
Input:
operando      Int
```

Temp:

raiz	*Int*

Return:

F_Raiz	*Int*

FB2 (Ordenación_calculo_salida)

Los parámetros declarados son:

Input:

matriz_ordenada	*Array[0..15] of Int*
matriz_calculo	*Array[0..15] of Struct*
direccion	*Int*

Temp:

dato_tempo_intercamb	*Int*
i	*Int*
j	*Int*
dato_desplaz	*Byte*

Programa

```
(*Ordena la matriz con el método burbuja.*)
FOR #j := 0 TO 14 DO
  FOR #i := #j TO 14 DO
    IF #matriz_ordenada[#j]< #matriz_ordenada[#i+1] THEN
      #dato_tempo_intercamb := #matriz_ordenada[#j];
      #matriz_ordenada[#j] := #matriz_ordenada[#i + 1];
      #matriz_ordenada[#i + 1] := #dato_tempo_intercamb;
    END_IF;
  END_FOR;
END_FOR;
(*Realiza los cálculos y los deja en una matriz de tipo STRUCT.*)
FOR #i := 0 TO 15 DO
  #matriz_calculo[#i].cuadrado := "F_Cuadrado"(#matriz_ordenada[#i]);
  #matriz_calculo[#i].raiz := "F_Raiz"(#matriz_ordenada[#i]);
END_FOR;
(*Procede a extraer el dato validándolo mediante la entrada E1.7
Previamente se ha debido seleccionar la dirección.*)
#dato_desplaz := "marca_desplaz";
```

```
IF "valida_salida" THEN
  IF "seleccion_vmedido_vcalculado" THEN
    "marca_salida" := #matriz_ordenada[#dato_desplaz];
  ELSE
    IF "seleccion_raiz_cuadrado" THEN
      "marca_salida" := #matriz_calculo[#dato_desplaz].raiz;
    ELSE
      "marca_salida" := #matriz_calculo[#dato_desplaz].cuadrado;
    END_IF;
  END_IF;
END_IF;
```

Para visualizar los resultados, se debe abrir el DB (Entrada_datos_DB) y la tabla de observación de variables. Poniendo online el DB, se pueden observar los valores de las diferentes matrices. Desde la tabla de observación y forzado se observa el resultado final deseado en la marca de salida (MW0).

AUTOMATIZACIÓN DE VARIOS MOTORES

Se debe realizar un programa secuencial en el que arrancan cuatro motores condicionados a la marcha. Los motores deben quedar en marcha hasta que estén todos arrancados y se active el pulsador de paro Pp. El Motor1 arranca con el pulsador P1, el Motor2 con el pulsador P3, el Motor3 con P3 y el Motor4 con P4.

Se va a realizar el ejercicio usando el método grafcet (Figura 213).

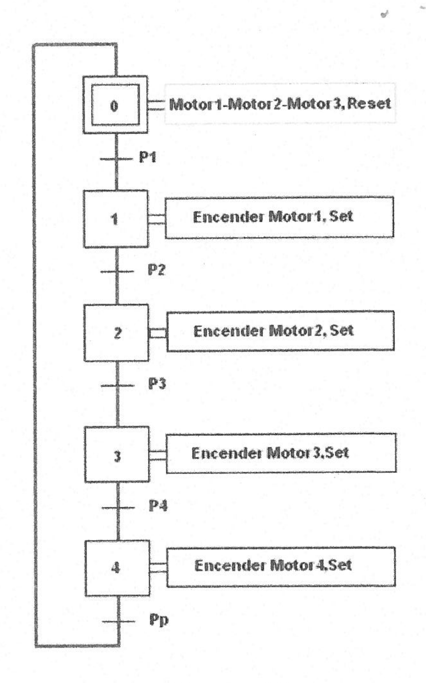


Figura 213

Se va a emplear un FB con parámetros declarados como estáticos. El OB1 solo tendrá la llamada a dicha FB.

OB1

El programa OB1 solo realiza la llamada al bloque de función.

"Función1"();

FB1, DB1 con parámetros estáticos

En el FB1 es donde se realiza el programa. Este programa tiene la característica de utilizar parámetros estáticos. Solo emplea como variable global la marca M0.0, que es la que se va a inicializar en el OB100 para que arranque el grafcet de una forma automática.

Los parámetros son e0, e1, e2, e3, e4 para cada una de las etapas del grafcet. Estos parámetros son variables locales y, por lo tanto, solo se pueden utilizar dentro de la propia FB.

Se declaran variables globales mediante la tabla de variables del PLC. Como son globales, se pueden utilizar en cualquier bloque de programación. Estas variables son los pulsadores:

E0.0------ P1 // E0.1------ P2 // E0.2 ------- P3 // E0.3 ------ P4 // E0.4 ------ Pp

También se incluye la marca que sirve para inicializar el grafcet, que es la M0.0 y se le ha denominado Etapa0.

#e0 := "Etapa0";	// La marca M0.0 (Etapa0), que se pone a uno en el OB100, se asigna al parámetro e0, que corresponde a la etapa 0.
IF "pm1" AND #e0 THEN #e0 := 0; "Etapa0" := 0; #e1 := 1; END_IF; IF "pm2" AND #e1 THEN #e1 := 0; #e2 := 1; END_IF; IF "pm3" AND #e2 THEN #e2 := 0; #e3 := 1; END_IF; IF "pm4" AND #e3 THEN #e3 := 0; #e4 := 1; END_IF;	// Aquí comienzan las transiciones. En cada IF se debe colocar como condición el estado de la etapa anterior y el pulsador que debe estar activado para que arranque el correspondiente motor. En esta primera transición se debe poner también a cero la variable Etapa0; si no, cada vez que empezase un ciclo, el valor de e0 sería 1.
IF "pp" AND #e4 THEN #e4 := 0;	// En la última transición se pone a uno la marca M0.0 (Etapa0) en vez del parámetro e0, ya que en la primera orden se igualan ambas.

"Etapa0" := 1; END_IF; IF #e1 THEN "motor1" := 1; END_IF; IF #e2 THEN "motor2" := 1; END_IF; IF #e3 THEN "motor3" := 1; END_IF; IF #e4 THEN "motor4" := 1; END_IF; IF #e0 THEN "motor1" := 0; "motor2" := 0; "motor3" := 0; "motor4" := 0; END_IF;	 // Aquí comienzan las acciones.

OB100

En AWL se coloca la marca M0.0 a uno para arrancar el grafcet.

```
SET
S   "Etapa0"
```

SEMÁFORO

Se trata de realizar un programa utilizando el lenguaje SCL. Este ejercicio ya se había planteado para su realización en el tema correspondiente al grafcet. El programa es la realización de un semáforo de peatones. El semáforo para los coches está normalmente verde y el de peatones en rojo hasta que un peatón desea cruzar y acciona el pulsador P. En este momento se desencadena la siguiente secuencia:

Al cabo de 10 s de pulsar P, el semáforo de coches pasa a ámbar.

Se mantiene en ámbar durante tres segundos, después de los cuales se enciende la luz roja, que permanece encendida 11 s.

A la vez se enciende el verde para peatones.

Está en verde para peatones durante 8 s fijo y 3 s más intermitente a una frecuencia de 0,5 s.

El semáforo vuelve a la posición normal, es decir, luz verde para coches y roja para peatones.

Al ser una automatización claramente secuencial, se realiza mediante grafcet. En la Figura 214 puede verse el grafcet.

Se utilizan variables globales. Estas son las marcas para las etapas y se declaran en la tabla de variables del PLC. El programa se va a escribir todo en el OB1. Habrá un OB100 para inicializar la etapa 0.

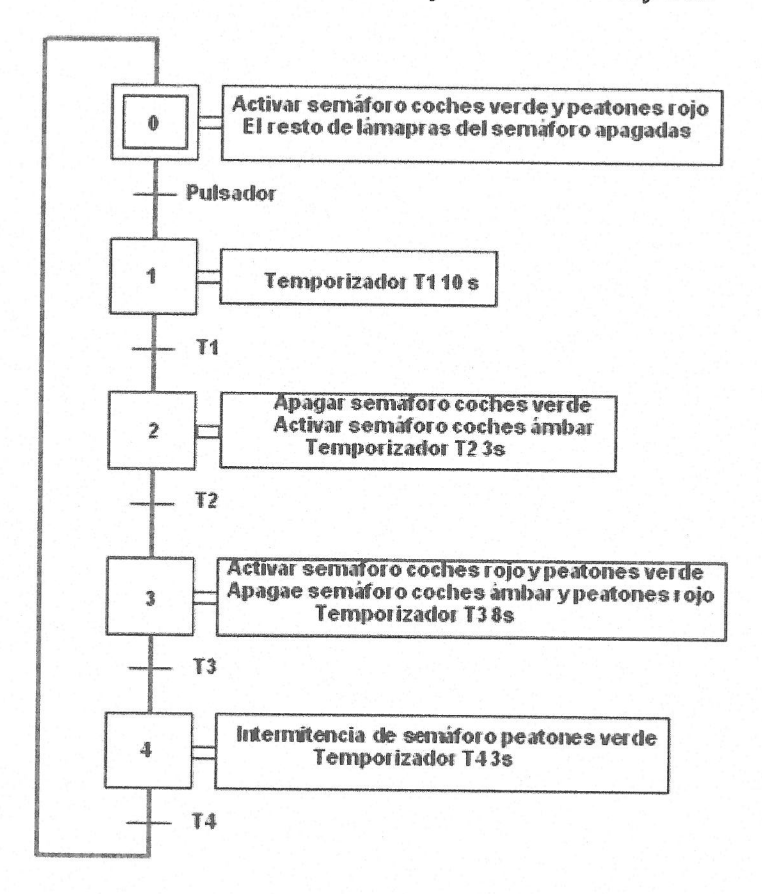


Figura 214

La variables globales (variables PLC) utilizadas y declaradas son:

Para cada una de las etapas:

m0..... M0.0 // m1 ..... M0.1 // m2 ..... M0.2 // m3 .... M0.3 // m4 ..... M0.4

Para las salidas:

ScV......Semáforo coche verde.....A0.0

ScR......Semáforo coche rojo..... A0.1

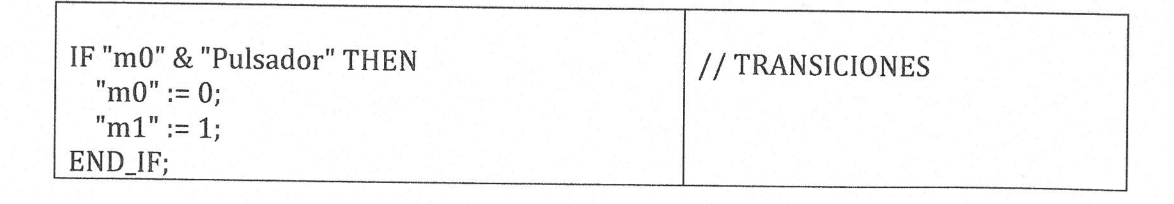
ScA......Semáforo coche ámbar..... A0.2

SpV......Semáforo peatones verde..... A0.3

SpR.......Semáforo peatones rojo.... A0.4

También se utilizan cuatro temporizadores IEC tipo TON y una marca de ciclo, MB20.

OB1



```
IF "m0" & "Pulsador" THEN                  // TRANSICIONES
  "m0" := 0;
  "m1" := 1;
END_IF;
```

IF "m1" & "IEC_Timer_0_DB".Q THEN "m1" := 0; "m2":= 1; END_IF; IF "m2" & "IEC_Timer_0_DB_1".Q THEN "m2" := 0; "m3":= 1; END_IF; IF "m3" & "IEC_Timer_0_DB_2".Q THEN "m3" := 0; "m4" := 1; END_IF; IF "m4" & "IEC_Timer_0_DB_3".Q THEN "m4" := 0; "m0" := 1; END_IF;	
IF "m0" THEN "ScV" := 1; "SpR" := 1; "ScR" := 0; "ScA" := 0; "SpV" := 0; END_IF; IF "m2" THEN "ScV" := 0; "ScA" := 1; END_IF; IF "m3" THEN "ScA" := 0; "SpR" := 0; "SpV" := 1; "ScR" := 1; END_IF; IF "m4" THEN "SpV" := "MarcaCiclo0,5sg"; END_IF;	//ACCIONES
"IEC_Timer_0_DB".TON(IN := "m1", PT := t#10s); "IEC_Timer_0_DB_1".TON(IN := "m2", PT := t#3s); "IEC_Timer_0_DB_2".TON(IN := "m3", PT := t#8s); "IEC_Timer_0_DB_3".TON(IN := "m4", PT := t#3s);	// Selección de los temporizadores.

OB100

Escrito en AWL.

```
SET
S   "m0"
```

CINTAS TRANSPORTADORAS

Se dispone de tres cintas transportadoras dispuestas como indica la figura. Por las cintas transportadoras van a circular cajas grandes y pequeñas indistintamente. El tamaño de las cajas es detectado por dos sensores. Para cajas grandes, se activan los sensores S2 y S3, y para las pequeñas solo S3. La cinta 3 dispone de otro sensor (S4), que detecta que la caja (pequeña o grande) ya está depositada en esa cinta.

El funcionamiento del sistema debe ser el siguiente:

Cuando se active el pulsador de marcha, queremos que arranque la cinta n.º 1.

Cuando llegue la primera caja a la cinta n.º 2 (sensor S1), se debe parar la cinta n.º 1 y poner en marcha la cinta n.º 2.

En la cinta n.º 2 se detecta si la caja es grande o pequeña. Si es grande, se debe poner en marcha la tercera cinta hacia arriba y, si es pequeña, la tercera cinta hacia abajo.

La cinta n.º 2 se para cuando la caja ya esté en la cinta n.° 3. La cinta n.º 3 se para a los 10 s de haberse puesto en marcha.

A continuación se pone en marcha de nuevo la primera cinta y vuelve a comenzar el ciclo sin necesidad de pulsar el pulsador de marcha. Esto se repite hasta que se active el paro, momento en que termina el ciclo y ya no se pone en marcha hasta volver a accionar la marcha.

Se deben contar las cajas grandes y las pequeñas. Cuando se llegue a 10 cajas grandes o 15 pequeñas, se indicará con sendas lámparas indicativas.

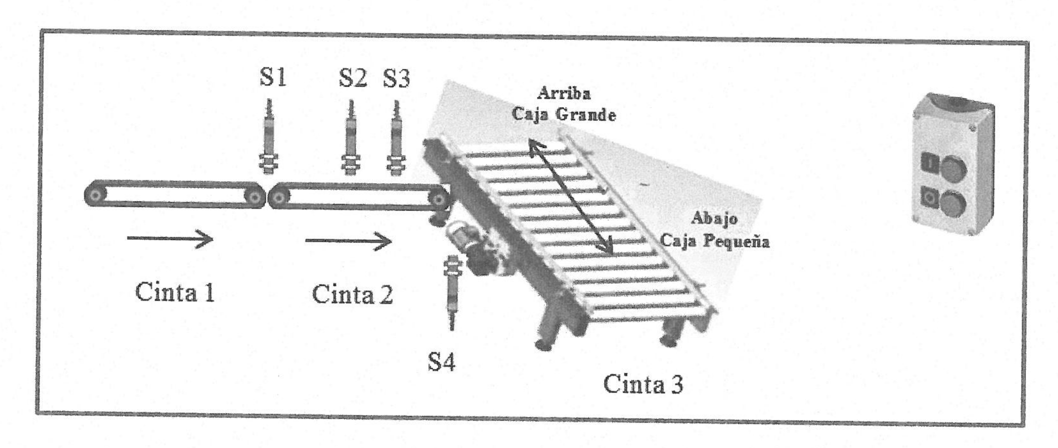


Figura 215

Se va a utilizar un FB donde se declararán las diferentes etapas del grafcet como parámetros estáticos.

Las variables globales (variables del PLC) y las variables locales (parámetros del FB) son las siguientes:

Variables Locales	Variables Globales	
etapa0	Etapacero	M10.0
etapa1	Ciclocontinuo	M10.1
etapa2	Cinta1	A0.0
etapa1.0	Cinta2	A0.1

etapa2.0 etapa3 etapa4	Cinta3 arriba	A0.2
	Cinta3 abajo	A0.3
	Marcha	E0.0
	Paro	E0.1
	Sensor1	E0.2
	Sensor2	E0.3
	Sensor3	E0.4
	Sensor4	E0.5
	AvisoCajasGrandes	A0.4
	AvisoCajasPequeñas	A0.5

La variable Ciclocontinuo se activa al accionar el pulsador de marcha y se desactiva al pulsar el paro. De esta forma, mientras no se activa el paro, el ciclo se repite sin necesidad de accionar el pulsador de marcha de una manera continua.

En la Figura 216 puede verse el grafcet. En este caso se abre una bifurcación *divergente O* y luego entra en una *convergencia O.*

Además, se utilizan dos contadores para contar las cajas grandes y las cajas pequeñas. Una salida se activa cuando las cajas grandes llegan a 10 y otra cuando las cajas pequeñas son 15.

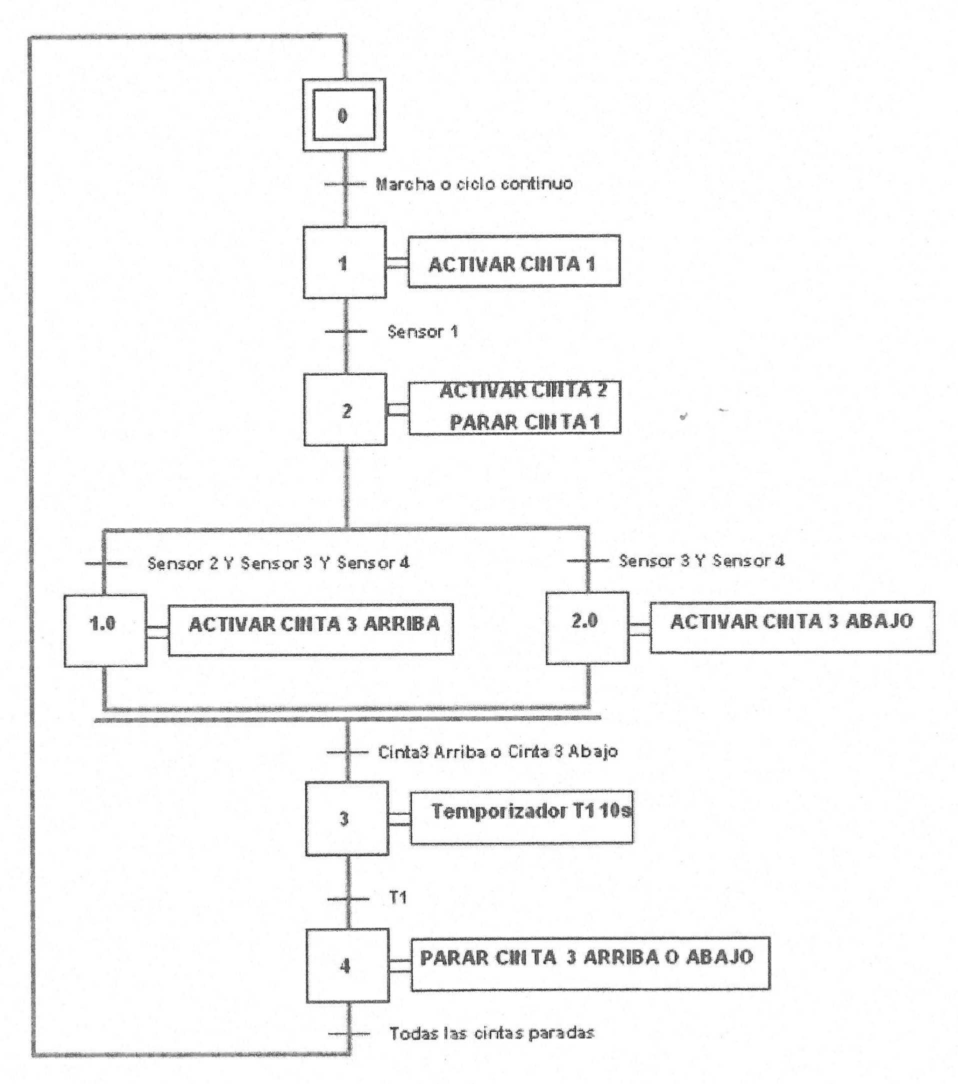


Figura 216

OB1

"FB1_DB_1"();

FB1

```
IF ("Marcha" OR "Ciclocontinuo") AND "Etapacero" THEN      // TRANSICIONES
  "Etapacero" := 0;
  #etapa1 := 1;
END_IF;
IF "Sensor1"AND #etapa1 THEN
  #etapa1 := 0;
  #etapa2 := 1;
END_IF;
IF "Sensor2" AND "Sensor3" AND "Sensor4" AND #etapa2
THEN                                                        // Caja grande
  #etapa2 := 0;
  #"etapa1.0" := 1;
END_IF;
IF "Sensor3" AND "Sensor4" AND #etapa2 THEN
  #etapa2 := 0;                                             // Caja pequeña
  #"etapa2.0" := 1;
END_IF;
IF ("Cinta 3abajo" OR "Cinta 3arriba") AND
(#"etapa1.0"OR #"etapa2.0") THEN
  #"etapa1.0" := 0;
  #"etapa2.0" := 0;
  #etapa3 := 1;
END_IF;
IF #etapa3 AND "T1".Q THEN
  #etapa3 := 0;
  #etapa4 := 1;
END_IF;
IF NOT "Cinta1" AND NOT "Cinta 2" AND NOT "Cinta
3abajo" AND NOT "Cinta 3arriba" THEN
  #etapa4 := 0;
  "Etapacero" := 1;
END_IF;
IF #etapa1 THEN
  "Cinta1" := 1;                                            // ACCIONES
ELSE
  "Cinta1" := 0;
END_IF;
IF #etapa2 THEN
  "Cinta 2" := 1;
ELSE
  "Cinta 2" := 0;
END_IF;
IF #"etapa1.0" THEN
  "Cinta 3arriba" := 1;
END_IF;
```

IF #"etapa2.0" THEN "Cinta 3abajo" := 1; END_IF; IF #etapa4 THEN "Cinta 3abajo" := 0; "Cinta 3arriba" := 0; END_IF; "T1".TON(IN:=#etapa3, PT:=t#10s);	
"Contador_Cajas_Grandes".CTU(CU:=#"etapa1.0", PV:=10, Q=> AvisoCajasGrandes); "Contador_cajas_pequeñas".CTU(CU:=#"etapa2.0", PV:=15,Q=> AvisoCajasPequeñas);	// Contadores
IF "Paro" THEN "Ciclo continuo" := 0; END_IF;	// Si se pulsa el paro, terminará el ciclo y no seguirá si no se pulsa marcha.
IF "Marcha" THEN "Ciclo continuo" := 1; END_IF;	// Si se pulsa la marcha, los siguientes ciclos empezarán de forma automática, sin necesidad de accionar el pulsador de marcha.

OB100

El OB100 solo inicializa el grafcet:

```
SET

S   "Etapacero"
```

SIMULADOR TIA PORTAL (v13) PARA EL PLC 1500

El simulador de PLC S7-1500 es diferente al del PLC S7-300. En este apartado estudiaremos la forma de proceder para poder simular los programas de estos autómatas.

Se abre el simulador accionando el icono . Si no está resaltado y no se puede activar, hay que tocar con el ratón sobre el PLC. Se abrirá una ventana como indica la Figura 217. A continuación aparece la ventana para comunicar con el PLC y se deben seleccionar las diferentes interfaces del simulador como indica la Figura 218. Seleccionar la búsqueda y cargar en el PLC simulado.

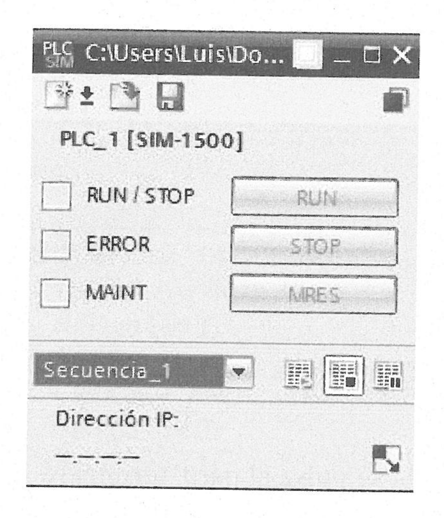


Figura217

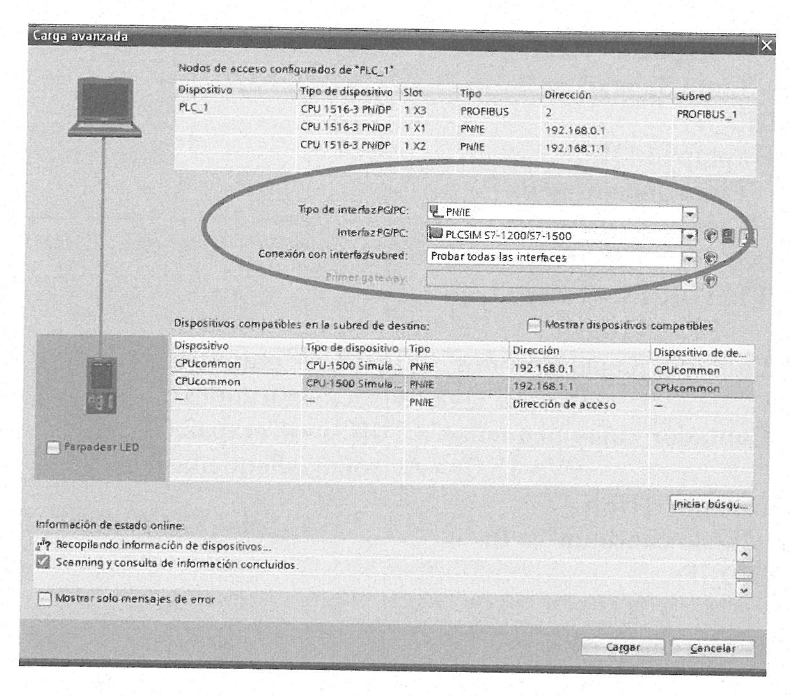


Figura 218

Una vez cargada la configuración en el PLC, la consola del simulador ya indica la dirección del PLC que se va a simular, tal como se ve en la Figura 219.

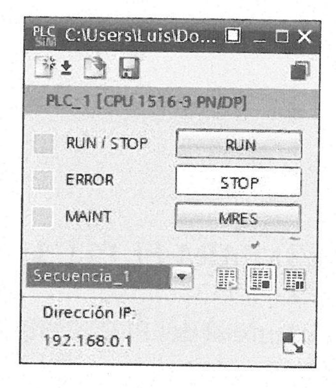


Figura 219

Para continuar con la configuración del simulador, se debe expandir accionando el icono ⬊. Aparecerá la pantalla de la Figura 220. Se debe pulsar en **Tablas SIM**.

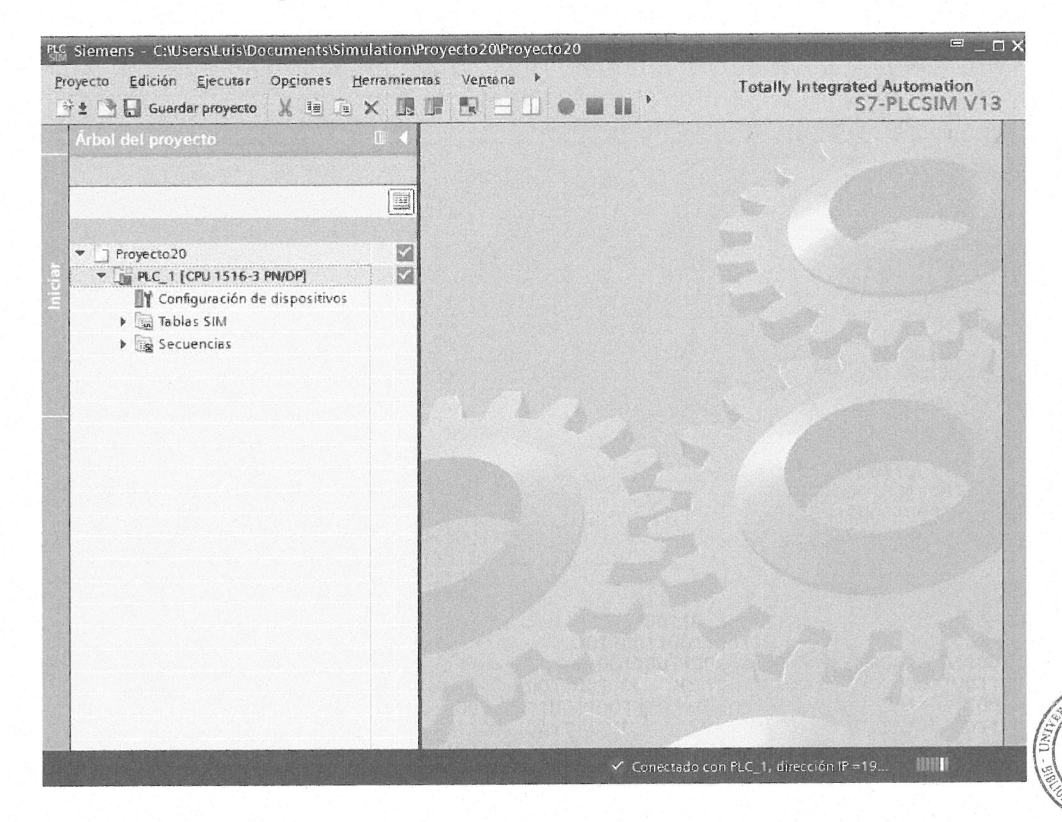


Figura 220

Una vez dispuestas todas aquellas variables que se deseen simular, solo hay que arrancar el PLC si no lo estaba ya, y **activar los bits** o cargar los **valores deseados**. Solo se podrán activar entradas. Para activar salidas se debe **permitir el forzado de salidas.** Si se ha permitido el forzado de salidas, hay que marcar la opción **Forzar coherente** y ejecutar el forzado que se realiza con el **icono del rayo**. Aparece un aviso con una exclamación en amarillo. Esto se debe hacer con el PLC parado porque si el programa utiliza alguna de estas salidas, no se podrá forzar.

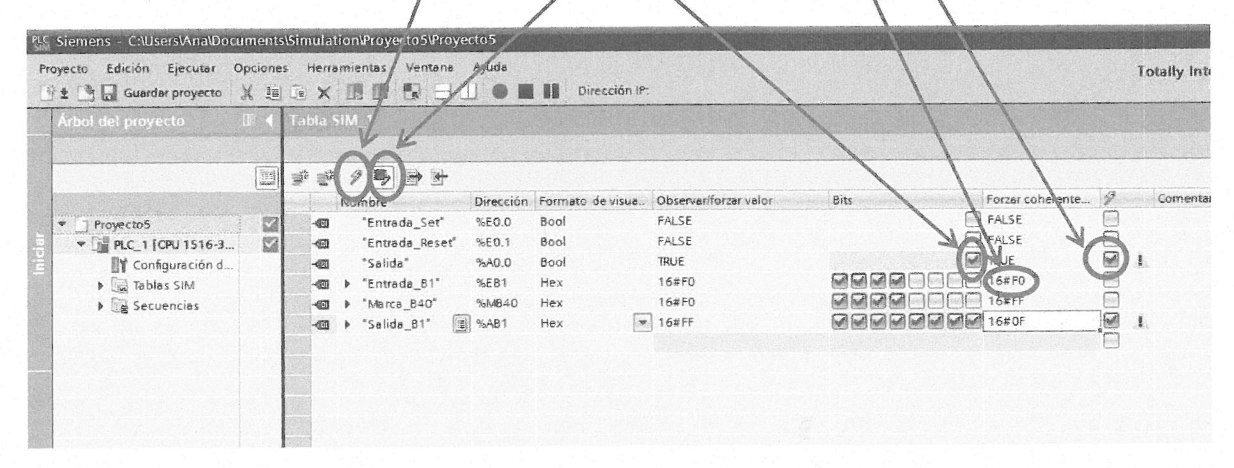


Figura 221



Son tiempos de cambios en la industria y, por lo tanto, en los sistemas automáticos. El PLC como parte fundamental de la automatización se prepara para la Industria 4.0. La digitalización, la integración y la nube van a ser parte de esa nueva industria.

Siemens apuesta fuerte por la Industria 4.0 y con el nuevo PLC 1500 lidera la iniciativa de esta nueva aventura. Se trata de un autómata que Siemens lanzó al mercado en el año 2013 y que ahora está experimentando una gran introducción en el mercado industrial mundial.

Muchos técnicos sienten cierto temor por la transición de los viejos sistemas a los nuevos. En este caso, el avance no es complicado y este libro ayuda a que esa evolución sea muy fácil, guiando de una forma clara, paso a paso, al interesado.

Se estudia el uso de la plataforma de programación de Siemens TIA PORTAL y se programa en lenguaje textual (AWL). Como las corrientes actuales en la programación van orientadas hacia el uso de lenguajes universales, una parte del libro está dedicada a la programación SCL, basada en la norma IEC 61131. Conseguir unificar la programación de cualquier fabricante de PLC's es un objetivo que se consigue bajo esta norma. Existen muchos libros sobre programación de PLC's pero no hay mucha bibliografía que se dedique de forma clara y sencilla a la programación SCL.

Este libro va dirigido a aquellas personas que quieran iniciarse en el conocimiento de la programación de los autómatas y también para aquellas que, conociendo el actual S7-300 quieran evolucionar hacia un futuro en el que, sin lugar a dudas, estará el PLC 1500 de SIEMENS.





www.marcombo.com

